Computación de Altas Prestaciones en Genómica

Héctor Martínez Pérez

Directores: **Sergio Barrachina Mir**

María Isabel Castillo Catalán

Castellón de la Plana, Octubre de 2019

Programa de Doctorado en Informática

Escuela de Doctorado de la Universitat Jaume I

Computación de Altas Prestaciones en Genómica

Memoria presentada por Héctor Martínez Pérez para optar al grado de Doctor por la Universitat Jaume I

Héctor Martínez Pérez **Sergio Barrachina Mir** **María Isabel Castillo Catalán**

Castellón de la Plana, Octubre de 2019

Financiación recibida

Los fondos de financiación para llevar a cabo la tesis doctoral provienen de la beca con referencia **BES-2015-072741** del Ministerio de Ciencia, Innovación y Universidades del Gobierno de España. Esta beca está asociada al proyecto con referencia **TIN2017-82972-R** titulado: **Técnicas algorítmicas para computación de alto rendimiento consciente del consumo energético y resistente a errores**.

Índice general

Índice de figuras

Índice de cuadros

Resumen

El ser humano está formado por millones de células. Estas células, a su vez, constan de diferentes partes. De todas estas partes, una de las más importantes es el núcleo, donde se localiza la información genética. En el interior del núcleo se encuentran 23 pares de cromosomas. El conjunto de estos cromosomas constituyen el ácido desoxirribonucleico, más conocido como ADN. Cada uno de los cromosomas está compuesto por una doble hélice de aminoácidos, que reciben el nombre de nucleótidos. Además, los aminoácidos que forman el ADN se agrupan formando conjuntos de genes que se copiarán en el ARN, o ácido ribonucleico. Finalmente, gracias al ARN se ensamblarán las moléculas de proteínas que formarán los diferentes órganos de nuestro cuerpo.

Gran parte de los estudios genómicos se centran en el estudio del ADN/ARN, ya que estas estructuras contienen la mayor parte de la información genética necesaria para el funcionamiento de nuestro organismo. Estos estudios genómicos consisten en extraer muestras de ADN/ARN de un organismo y secuenciarlas, obteniendo así una multitud de pequeñas secuencias. La tarea de secuenciación se lleva a cabo mediante secuenciadores. Estas máquinas generan ficheros que contienen millones de cadenas de texto de un tamaño de entre 50 y 400 caracteres (o incluso mayores). Estos caracteres codifican cada uno de los aminoácidos: A (adenina), C (citosina), G (guanina) y T (tinina).

Una vez se han generado los ficheros con las secuencias, el siguiente paso es procesarlos. Este procesamiento consiste en buscar y encontrar cada una de estas secuencias en un genoma de referencia y almacenar los resultados en un fichero de salida. Esta tarea es muy compleja ya que el genoma de referencia, en el caso del genoma humano, ocupa aproximadamente 3 GiB. Por otro lado, estas secuencias, ya sea por variaciones genéticas naturales o por enfermedades, pueden llegar a contener multitud de errores, lo que complica la búsqueda. Además, en el caso del ARN, como está formado por fragmentos de ADN separados entre sí, muchas de las secuencias se habrán generado a partir de fragmentos no contiguos del genoma de referencia, lo que hace más difícil su localización.

Cuando ya se han generado los resultados del alineamiento, el estudio continua con el análisis de las variantes encontradas. Mediante este estudio, se intenta buscar una relación entre las variantes encontradas y las enfermedades o características que presenta el individuo.

Debido a la continua evolución de las máquinas para la secuenciación de ADN/ARN, la cantidad de datos generados se ha incrementado de forma sustancial en los últimos años. Esto ha conllevado la necesidad de disponer de software más eficiente para el procesamiento de este tipo de datos, que haga posible avanzar más rápido en el estudio de enfermedades tales como el cáncer, la diabetes, el trastorno bipolar, etc.

El trabajo realizado durante la tesis doctoral se ha dividido en las siguientes dos partes:

- La primera de ellas ha consistido en el desarrollo de un software eficiente para el alineamiento de secuencias de ADN/ARN, capaz de superar tanto la sensibilidad como la especificidad de los alineadores actuales, a la vez que se reduce el tiempo de procesamiento. Para llevar a cabo esta tarea se han usado diferentes técnicas para la búsqueda y alineamiento de estas secuencias, tales como: la transformada de Burrows-Wheeler (BWT) [5], la utilización de una colección de sufijos (SA) [30] y el algoritmo de Smith-Waterman (SWA) [46]. Además de estas técnicas algorítmicas, también se han usado otras técnicas HPC (*High Performance Computing*) para acelerar el procesamiento. Esto permite ejecutar el alineador en plataformas multiprocesador y clúster aprovechando todos los recursos del sistema. Como resultado del trabajo realizado se ha conseguido desarrollar un alineador para secuencias de ADN/ARN basado en BWT y SA, además de una versión para sistemas clúster del alineador para secuencias de ARN. Por otro lado, se ha llevado a cabo la implementación de un *framework* capaz de ejecutar cualquier alineador de ADN/ARN en un entorno clúster, acelerando de esta manera su procesamiento de forma transparente para el usuario. Este *framework* permite al usuario añadir una segunda fase de refinamiento de datos para mejorar la sensibilidad del procesamiento.

- La segunda parte del trabajo se ha centrado en el estudio de la epistasis. Esta ocurre cuando uno o varios genes interactúan con otros ocultando o potenciando su efecto. Este estudio se lleva a cabo una vez se han detectado las variantes de el o los individuos mediante el alineamiento de las secuencias de ADN/ARN. En este caso, el estudio se ha centrado en la optimización de la aplicación FaST-LMM, una de las más conocidas y usadas para la detección de la epistasis. Esta aplicación estudia todas las posibles combinaciones de variantes de dos en dos. Este tipo de procesamiento conlleva multitud de cálculos y, por lo tanto, su procesamiento es muy costoso. Por estos motivos se ha seleccionado esta aplicación para optimizarla la medida de lo posible. Así pues, el trabajo ha consistido en un estudio detallado del rendimiento del módulo de Fast-LMM que estudia la epistasis. A continuación, se ha desarrollado una versión donde se optimizan aquellas partes de la aplicación que generaban cuellos de botella. En esta parte también se ha implementado una versión que incluye la utilización de un acelerador gráfico de tipo GPU para descargar los cálculos matriciales que se requieren en el algoritmo a este tipo de dispositivos. Finalmente, se ha desarrollado una versión para sistemas clúster que permite distribuir el fichero de entrada entre los nodos del sistema y así permitir que los nodos del clúster colaboren en el procesamiento. En general, estas implementaciones han permitido reducir notablemente el tiempo de procesamiento y, en el último caso, trabajar con ficheros de entrada más grandes obteniendo la misma precisión en los resultados.

Agradecimientos

Después de muchos años de dedicación, esfuerzo y perseverancia como miembro del grupo de investigación HPC&A de la Universidad Jaume I, además de la colaboración con el Centro de investigación Principe Felipe y el grupo de biología de la Universidad Pompeu Fabra, se ha conseguido culminar el trabajo propuesto en esta tesis doctoral. Gracias a todos ellos por el apoyo ofrecido durante todo este tiempo.

Gracias a la Universidad Jaume I, a la Comisión Europea y al Ministerio de Ciencia y Educación, ya que sin su ayuda económica este trabajo no hubiera sido posible.

Mis más sinceros agradecimientos a Sergio Barrachina Mir, María Isabel Castillo Catalán y Enrique S. Quintana Ortí por ofrecerme siempre las máximas facilidades para trabajar, por ser mi principal apoyo durante el trabajo desempeñado y por ser unos excelentes compañeros/amigos durante todo este tiempo. Además, también me gustaría agradecer a Sergio Barrachina y María Isabel Castillo el haber dedicado tanto tiempo y esfuerzo en la revisión de esta tesis. Finalmente, agradecer sobre todo a María Isabel Castillo el trato que ha tenido conmigo durante todos estos años, ya que hemos hablado día sí y día también y ha sido un gran apoyo tanto en el ámbito laboral como personal, por todo esto, ¡muchas gracias!

También me gustaría agradecer a Rafael Mayo Gual su tutorización y el ayudarnos con su experiencia y con sus ideas tan originales para resolver algunos de los problemas afrontados.

Además, me gustaría agradecer especialmente a mis compañeros de trabajo desde el inicio de esta andadura, ya que hicieron que cada día fuera una alegría el ir a trabajar. Ellos son: Sergio Iserte, Sandra Catalán y Adrián Castelló. Además de a todos ellos, gracias igualmente a Francisco José Clemente, quien me ha acompañado y ayudado desde el inicio de la carrera hasta casi el final de la tesis con sus conocimientos.

Gracias a las personas del grupo HPC&A de la Universidad Jaume I con las que he tenido más contacto y que en algún momento me han ayudado: Jose Ignacio Aliaga, Rafael Rodríguez, José Manuel Badía, Juan Carlos, Germán León, Rocio Carratalá y María Barreda.

Gracias a mis amigos y familia, en especial a mis padres, por haber estado siempre a mi lado ayudándome y apoyándome en los momentos más difíciles.

Y por supuesto, un millón de gracias a mis dos principales pilares durante todo este tiempo: mi pareja, Esmeralda, y mi hijo, Izan. Ellos han hecho posible que cada día me levantara

con fuerza, ilusión y ánimo para seguir trabajando. Ambos son una fuente inagotable de energía y siempre están a mi lado cuando los necesito.

¡Gracias a todos!

Castellón, Octubre de 2019.

CAPÍTULO 1

Introducción general

En este capítulo se introduce el trabajo realizado en esta tesis doctoral. En primer lugar, se describen de forma general algunos de los principales problemas relacionados con la bioinformática. A continuación, se exponen brevemente los tres problemas bioinformáticos que se han abordado en esta tesis. Finalmente, se detallan los objetivos que se pretende alcanzar.

1.1 Biología e informática

En los años 50 del siglo XX se dieron dos hechos cuya evolución posterior ha dado lugar a grandes avances en biología y en el desarrollo de nuevos tratamientos en medicina:

- El primero fue la publicación, en 1953, del artículo «Molecular Structure of Nucleic Acids: A Structure for Deoxyribose Nucleic Acid» [55] donde se describe por primera vez la estructura del ADN. La definición de esta estructura se puede considerar como el inicio de la genómica.
- El segundo se sitúa el 2 de diciembre de 1954. Ese día se realizó la demostración formal del *Naval Ordenance Research Calculator* (NORC) construido por IBM en el *Watson Scientific Computing Laboratory*. El NORC se considera el primer supercomputador y fue el sistema con más capacidad de cálculo entre 1954 y 1963. Se puede decir que, con la construcción de este computador, empezó la disciplina de la supercomputación.

Durante los siguientes 20 años, la biología y la informática fueron evolucionando de forma separada, hasta que alrededor de 1970 comenzó a aplicarse la computación para la resolución de problemas de biología, dando lugar al nacimiento de lo que hoy llamamos bioinformática. A este proceso contribuyó notablemente, a partir de 1990, el desarrollo del proyecto del genoma humano. Sin embargo, ha sido sobre todo en los últimos años cuando se ha producido una tremenda expansión en el campo de la bioinformática, motivada principalmente por el auge de una nueva tecnología de secuenciación de ADN y ARN, conocida como secuenciación de próxima generación (NGS). Esta nueva técnica ha permitido, por un lado, reducir considerablemente el tiempo necesario para secuenciar un genoma, de años a tan solo unos días y, por otro lado, disminuir los costes del secuenciamiento. Esto ha provocado que el análisis genético de alto rendimiento sea cada vez más accesible a investigadores y médicos. Por contra, la aplicación de estas nuevas tecnologías ha generado un

aumento exponencial de la cantidad de información a procesar y, a medida que este incremento se ha hecho más notable, el análisis computacional se ha convertido en uno de los factores limitantes en la investigación genómica, tanto desde el punto de vista de las necesidades de almacenamiento, como de las de cálculo. De hecho, en cuanto a la cantidad de datos generados que deben procesarse, la genómica está a la misma altura que los grandes experimentos de física fundamental o de observación astronómica. De hecho, algunas predicciones señalan que en 2025 la genómica requerirá el almacenamiento y la gestión de entre 2 y 40 Exabytes por año y cientos de millones de horas de CPU [47].

En general, la gran mayoría de estudios bioinformáticos engloban un flujo de trabajo que comienza con la obtención del conjunto de las secuencias a analizar (secuenciación) y finaliza con los resultados del análisis que permiten detectar las variantes genéticas presentes en un individuo. Este flujo pasa por varias etapas de procesamiento que, como se ha dicho, requieren de grandes capacidades de almacenamiento y de cálculo, siendo habitual que se ejecuten distintas aplicaciones para cada etapa.

El paso previo a este flujo de trabajo consiste en la obtención, mediante un instrumental específico denominado secuenciador, de las secuencias de bases de nucleótidos correspondientes a los trozos en los que se ha fragmentado previamente el ADN o ARN de la muestra a analizar. A estas secuencias se les denomina lecturas y la tecnología actual permite obtener lecturas entre 50 y 400 nucleótidos (nts), aunque su tamaño tiende a aumentar. El resultado de una secuenciación son cientos de millones o incluso miles de millones de lecturas que se almacenan en un fichero con un formato preestablecido (siendo FASTQ o FASTA los más utilizados). A partir de este punto, comienza el flujo de procesamiento de estas lecturas, que consta de tres pasos:

(i) El primer paso consiste en localizar las lecturas en un genoma de referencia para determinar de qué coordenadas genómicas provienen. Las aplicaciones que realizan esta tarea reciben el nombre de alineadores o mapeadores. Estos mapeadores se basan en diferentes técnicas y estructuras de datos. Las técnicas más utilizadas son la transformada de Burrows-Wheeler (BWT) [5], la utilización de un índice de texto completo en un espacio reducido (FMindex) [13], la utilización de una matriz de sufijos (SA) [30] y el algoritmo de Smith-Waterman (SWA) [46]. Este primer paso constituye un proceso muy costoso desde el punto de vista computacional. El resultado de este paso, los alineamientos de las secuencias, suele almacenarse en un fichero SAM (*sequence alignment/map*) o BAM (*binary alignment/map*) [26], que contendrá toda la información necesaria para localizar de forma exacta cada una de las lecturas originales en el genoma de referencia.

(ii) La siguiente fase del procesamiento consiste en detectar e identificar las variantes genéticas presentes en las muestras. Estas variantes pueden consistir en un cambio de unas bases por otras o en inserciones y borrados (*indels*). Estas variantes pueden ser debidas a enfermedades, a ruido genético sin ningún efecto funcional o a posibles errores de secuenciación. La dificultad de esta etapa se debe a tres factores: (1) la presencia de *indels* que constituye la mayor fuente de falsos positivos; (2) los errores procedentes de la secuenciación, sobre todo si las lecturas son pequeñas; y (3) la variabilidad en la calidad de las distintas partes de las secuencias, sobre todo en sus extremos. El ratio de falsos positivos y de falsos negativos es una preocupación constante durante esta fase. Por este motivo, es recomendable realinear las lecturas para mejorar la calidad mediante un algoritmo que permita aumentar la sensibilidad, para lo que se suele utilizar la inferencia Bayesiana, el test de probabilidad o el desequilibrio de ligamiento. Por otra parte, conviene tener en cuenta que, aunque la mayor parte de la investigaciones de enfermedades se centran en errores en los nucleótidos, la detección de *indels* es

igualmente importante. Muchos estudios han demostrado que pequeños *indels* están asociados con enfermedades tales como el Alzheimer, el autismo, la esquizofrenia, etc. Por lo tanto, los alineadores deben ser capaces de localizar correctamente aquellas secuencias que contengan *indels*. Una vez realizado el alineamiento, existen herramientas que permiten llevar a cabo este análisis. Entre estas podemos destacar: Pindel [56] y Dindel/GATK [40]. El resultado de esta segunda etapa genera una lista con las diferencias presentes entre el genoma analizado y el de referencia.

(iii) La última etapa del procesamiento consiste en realizar un filtrado de las variantes detectadas y determinar cuáles están relacionadas con el estudio que se esté realizando. En el caso de una enfermedad como el cáncer, un método común de filtrado consiste en eliminar las variantes que están presentes tanto en individuos sanos como en individuos con cáncer, y mantener únicamente las variantes que aparecen en los individuos con cáncer. Por el contrario, si se trata de un estudio genético enfocado a la herencia, el filtrado se basa en buscar los patrones diferentes que son comunes a los distintos genomas analizados. Además de los anteriores filtrados, también se pueden seleccionar variantes basadas en anotaciones de otros estudios, esto es, en conocimientos adquiridos en análisis realizados anteriormente o en efectos funcionales predichos. Existen diversas aplicaciones que permiten llevar a cabo estos filtrados, entre las que se encuentran: SIFT [43], PolyPhen [2], VariBench [42] y snpEFF [8].

1.2 Problemas abordados en esta tesis

Esta tesis se centra en tres problemas muy concretos del campo de la genética y en el desarrollo de sus correspondientes aplicaciones bioinformáticas. Estos tres problemas son: el alineamiento de secuencias de ADN y de secuencias de ARN contra un genoma de referencia, y el estudio de la epistasis. Tanto la secuenciación de ADN (ADN-sec) como la de ARN (ARN-sec) son ampliamente utilizadas debido a sus implicaciones clínicas. Concretamente, gracias al ADN-sec se pueden revelar las causas de las enfermedades raras y, en última instancia, mejorar la atención médica del paciente [4]. Por otro lado, gracias al ARN-sec es posible cuantificar la expresión de genes que son activados/reprimidos por una enfermedad, lo que ayuda a entender su etiología. Finalmente, una vez realizado el alineamiento de las lecturas, uno de los distintos tipos de estudios que se realizan a raíz de las variantes encontradas, es la detección de epistasis. Este estudio consiste en determinar qué variantes genómicas se interrelacionan entre sí en uno o en multitud de individuos, cuando estos presentan un determinado rasgo o enfermedad.

Así pues, la primera parte de nuestro trabajo se ha enfocado en el desarrollo de una aplicación capaz de alinear secuencias de ADN y de ARN contra un genoma de referencia. Como se ha comentado anteriormente, la gran cantidad de datos genómicos proporcionados por las nuevas tecnologías de secuenciación conlleva la necesidad de disponer de software capaz de explotar eficientemente los recursos de un sistema de computación. En esta línea, se ha desarrollado un alineador que utiliza diferentes técnicas para el alineamiento de secuencias de ADN/ARN. Este alineador se puede clasificar según dos criterios: las técnicas de alineamiento utilizadas y las técnicas de computación de altas prestaciones aplicadas en su implementación.

En función de la técnica de alineamiento empleada, el primer método desarrollado utiliza inicialmente la transformada de Burrows-Wheeler (BWT) para la búsqueda de secuencias de forma rápida y, a continuación, el algoritmo de Smith-Waterman para las lecturas más complejas que no hayan podido ser alineadas mediante la técnica anterior.

Se ha desarrollado un segundo método que, en lugar de la transformada de Burrows-Wheeler, utiliza una matriz de sufijos para realizar la primera criba. Al realizar este cambio, se ha

conseguido acelerar sustancialmente esta etapa, aunque a costa de un mayor consumo de memoria. En esta implementación también se han incorporado varias técnicas de refinamiento, lo que ha permitido obtener una mayor sensibilidad que la conseguida por otro alineadores.

En cuanto a las técnicas de computación de altas prestaciones, se han aplicado diferentes tipos de técnicas de optimización y de paralelización. Por un lado, se han desarrollado diversos métodos enfocados a la ejecución paralela en un solo nodo con múltiples procesadores y, por otro, un método orientado a su ejecución paralela en varios nodos con múltiples procesadores por nodo (que forman parte de un clúster). Para explotar los recursos del primer tipo de sistemas, el método de alineamiento se organiza en una secuencia de etapas, donde cada etapa lleva a cabo una tarea distinta. La ventaja de esta organización es que, como los datos de entrada son independientes unos de otros, se pueden procesar distintas etapas, que utilizan datos de entrada diferentes, en paralelo. Esta organización optimiza la utilización de los distintos núcleos de procesamiento que constituyen los sistemas multiprocesador actuales. Por otro lado, se ha diseñado un método que, además de explotar el paralelismo multiprocesador, es capaz de explotar el paralelismo a nivel de los distintos nodos de un clúster. Este método distribuye de forma equilibrada los datos de entrada entre todos los nodos de cómputo y hace que todos los nodos procesen en paralelo su parte.

Finalmente, para completar esta parte del trabajo, se ha desarrollado un alineador genérico (*framework*) para el alineamiento de secuencias de ADN/ARN, que permite que cualquier mapeador pueda ejecutarse en un clúster de computadores aprovechando todos sus recursos sin tener que modificar su código para ello y de forma totalmente transparente al usuario. Además, siguiendo el diseño de nuestras anteriores implementaciones, el *framework* se organiza en un flujo de trabajo de dos etapas, y permite que en cada una de ellas se pueda ejecutar un alineador distinto. Con esta configuración, por ejemplo, la primera etapa puede, idealmente, utilizar un alineador rápido, para procesaar rápidamente un gran número de lecturas, mientras que la segunda puede utilizar un alineador más sensible, y probablemente más lento, para actuar como una etapa de refinamiento sobre las lecturas no mapeadas por el primer alineador. De esta forma, se obtendría un alineador con una mayor sensibilidad que la del primer alineador y más rápido que el segundo. Además, no debemos olvidar que ambos alineadores se estarían ejecutando de forma paralela en un clúster (cuando no estaban necesariamente desarrollados para esto).

La segunda parte del trabajo se ha centrado en el estudio de la epistasis. Este análisis requiere un procesamiento muy complejo y costoso, y por lo tanto, es necesario disponer de un software optimizado para poder llevarlo a cabo en un tiempo razonable. Existe una gran cantidad de aplicaciones que realizan este tipo de estudios, pero una de las más utilizadas, por su alta sensibilidad, es FaST-LMM. Esta aplicación consigue una alta sensibilidad gracias a que evalúa todas las posibles combinaciones de parejas de variantes. El trabajo de esta tesis ha consistido en realizar un estudio detallado de los cuellos de botella que presenta esta aplicación. A partir de este análisis, se han realizado diferentes optimizaciones utilizando algunas técnicas de computación de altas prestaciones. Además, se ha desarrollado una nueva implementación que permite explotar los recursos de un clúster de computadores con procesadores gráficos.

1.3 Objetivos

Como se ha introducido en el apartado anterior, el objetivo general de esta tesis es el desarrollo de dos aplicaciones bioinformáticas con un fin muy específico: el alineamiento de secuencias de ADN/ARN y el análisis de la epistasis. Ambas aplicaciones utilizarán técnicas HPC para reducir el tiempo de procesamiento y deberán ofrecer una alta sensibilidad.

Este objetivo general se puede descomponer en los siguientes objetivos más específicos:

- Estudiar la documentación existente de alineadores actuales de secuencias de ADN/ARN.
- Desarrollar varios alineadores para secuencias de ADN/ARN que sean capaces de mejorar los resultados obtenidos por los alineadores actuales, tanto a nivel de sensibilidad como de coste temporal. Estos utilizarán diferentes técnicas de búsqueda de cadenas y de HPC.
- Desarrollar un alineador genérico, o *framework*, que permita al usuario, de forma transparente, ejecutar cualquier alineador actual sobre un clúster de computadores explotando todos los recursos de este tipo de sistemas de computación.
- Implementar un entorno de experimentación y validación de pruebas: definir las plataformas y arquitecturas, las herramientas de simulación de lecturas de ADN/ARN, los bancos de datos reales para las pruebas y analizar los resultados obtenidos.
- Estudiar la documentación existente de aplicaciones actuales para el estudio de la epistasis.
- Descubrir los principales cuellos de botella del módulo de FaST-LMM para el estudio de epistasis.
- Desarrollar un módulo optimizado para el estudio de epistasis basado en FaST-LMM utilizando técnicas HPC para reducir el tiempo de procesamiento.
- Desarrollar de una nueva implementación de este módulo de epistasis que pueda ejecutarse en un clúster de computadores con GPU.
- Implementar un entorno de experimentación y validación de pruebas: definir las plataformas y arquitecturas, seleccionar los bancos de datos reales y estudiar los resultados obtenidos.

Mediante este trabajo se pretende lograr un avance en el alineamiento de secuencias de ADN/ARN y en la detección de epistasis para que la comunidad científica disponga de mejores herramientas para el estudio de enfermedades y para otros estudios biológicos.

1.4 Metodología y desarrollo

En el desarrollo de esta tesis se ha empleado la metodología clásica de desarrollo de software enfocada al modelo en cascada, con las siguientes etapas:

- Revisión de la literatura actual y estudio de los requisitos.
- Diseño del algoritmo.
- Desarrollo de la aplicación.
- Validación de los resultados.
- Optimización y mantenimiento de la aplicación.

La tesis doctoral se ha desarrollado bajo las anteriores etapas. Asimismo, incluye información detallada sobre la documentación recopilada, el desarrollo de las aplicaciones realizadas, los resultados obtenidos y las conclusiones a las que se ha llegado. En general, la tesis doctoral se ha estructurado en capítulos, y a su vez, en dos partes fundamentales. La primera de estas partes pretende dar una visión global de las técnicas usadas para el desarrollo de un alineador eficiente de secuencias de ADN/ARN, y la segunda de ellas, la optimización del software para la detección de epistasis.

Parte I

Alineamiento de secuencias de ADN/ARN

CAPÍTULO 2

Introducción

Este capítulo introduce uno de los principales problemas abordados en esta tesis: el alineamiento de lecturas de ADN/ARN. El capítulo se divide en dos partes. En la primera, se describe brevemente en qué consiste el análisis completo de las secuencias de ADN/ARN, centrándonos en el aspecto que más nos interesa, el alineamiento de estas secuencias sobre un genoma de referencia. En la segunda parte, se describen algunas de las técnicas más utilizadas para su mapeo y se muestra un listado de las aplicaciones más conocidas para el alineamiento de secuencias de ADN/ARN.

2.1 El análisis del ADN/ARN

El ácido desoxirribonucleico, abreviado como ADN, es un ácido nucleico que contiene las instrucciones genéticas usadas en el desarrollo y funcionamiento de todos los organismos vivos y algunos virus; también es el responsable de la transmisión hereditaria. El ADN está organizado en forma de cromosomas. Estos cromosomas se duplican, antes de la división celular, para transferir la información de una célula a otra. En los organismos vivos, el ADN se presenta como una doble cadena de aminoácidos (doble hélice) conectadas entre sí mediante una serie de enlaces de hidrógeno. Estos aminoácidos son conocidos como nucleótidos y hay cuatro tipos diferentes: adenina (A), citosina (C), guanina (G) y timina (T). Estos nucleótidos se agrupan formando intrones y exones. Los exones, a su vez, se unen para formar los genes, que constituyen la unidad funcional y fundamental de la herencia.

Por otra parte, el ácido ribonucleico, abreviado como ARN, se forma agrupando y duplicando los exones del ADN por medio de un proceso denominado transcripción. A través del ARN se transfiere la información esencial para la fabricación de las proteínas. Éstas son las encargadas de proporcionar a las células todos los recursos necesarios para llevar a cabo su desarrollo funcional.

Tanto la secuenciación de ADN (ADN-sec) como la de ARN (ARN-sec) son utilizadas para un gran número de aplicaciones clínicas. Por ejemplo, para detectar las causas de enfermedades raras [21], personalizar el cuidado de cada paciente en función de su genoma [3] y entender la etimología de determinadas enfermedades cuantificando qué genes están activados/desactivados [45].

El análisis de ADN/ARN requiere de varias etapas, comenzando con la obtención de las secuencias que se desean analizar. Este primer paso conlleva la realización de varias operaciones: en primer lugar, se extrae la molécula de ADN/ARN, a continuación, se fragmenta y, mediante

una máquina de secuenciación se obtienen las secuencias de nucleótidos que forman las lecturas. La necesidad de secuenciar moléculas de ADN/ARN, en el menor tiempo y con la mayor precisión posibles, ha generado una revolución sin precedentes en el campo de la biología. Esto ha dado lugar al desarrollo de las tecnologías de secuenciación de segunda generación (NGS), que son capaces de secuenciar ADN y ARN con un coste reducido y una gran velocidad.

Por otra parte, la continua evolución de las tecnologías de secuenciación ha implicado un incremento notable del tamaño de las lecturas obtenidas. Hace unos años, estas lecturas rondaban los 50 nucleótidos (nts) y, actualmente, se están generando lecturas de 400 nts [7] e incluso superiores.

Lo anterior ha propiciado que el tamaño de los ficheros donde se almacenan las lecturas alcance tamaños superiores a 100 GiB. Estos ficheros siguen un formato preestablecido para que puedan ser manipulados en las posteriores etapas del análisis. Los dos formatos de fichero más utilizados son: Fasta y FastaQ [9]. Se diferencian en el número de líneas que utilizan para representar cada una de las lecturas y en el carácter de inicio. En la Figura 2.1 se muestra cómo se codifica una lectura en un fichero FastaQ, que es el formato más utilizado. Como se puede observar, para cada lectura se almacenan 4 líneas de texto: el identificador de la lectura, la secuencia de nucleótidos, un separador de linea y, por último, tantos caracteres en ASCII como nucleótidos hay en la línea 2. Los caracteres de la última línea codifican la calidad con la que se ha obtenido cada nucleótido, que va desde 0x21 (la calidad más baja; «!» en ASCII) hasta 0x7E (la calidad más alta; «~» en ASCII).

```
@seq.1a
AGATGCTCACATAAAGCTCTGAGACTCCCAAATGGGTGACATGCTCAGTG
+
<=<=>===>==========<==>=>=>=======<=<=<>==<=>=<>=<
```

Figura 2.1: Ejemplo de lectura en formato FastQ

Una vez se han obtenido las lecturas, el primer paso del análisis consiste en aplicar un filtro a estas lecturas. Para ello, se utiliza la calidad de las lecturas almacenada en el fichero. Aquellas que contienen una calidad muy baja son descartadas para que no añadan ruido al procesamiento. Muchos de los alineadores de lecturas ya tienen integrada esta fase del procesamiento, ahorrando así al usuario este primer paso.

La siguiente etapa del análisis consiste en localizar las coordenadas genómicas de cada lectura en un genoma de referencia [23]. El software que lleva a cabo esta tarea tiene que ser capaz de encontrar millones de lecturas en un genoma en el menor tiempo y con la mayor sensibilidad posibles. Debido a las características del proceso, este paso es computacionalmente complejo y muy costoso. Además, en el caso de ARN, la dificultad se incrementa porque éste está formado exclusivamente por exones (recordemos que el ADN está formado por exones unidos por intrones). Por lo tanto, una lectura de ARN puede estar formada por partes de varios exones, que en el genoma de referencia estarán separadas entre sí (hasta por 500.000 nucleótidos).

Por otro lado, debido a las mutaciones genéticas y fallos en la secuenciación, las lecturas generadas por los secuenciadores pueden contener múltiples errores. Éstos pueden aparecer en forma de cambios de bases, borrados o inserciones. Por este motivo, el proceso de mapeo de las secuencias debe tolerar estos errores.

El resultado final de esta etapa, tanto para el procesamiento de ADN como para el de ARN, es uno o varios ficheros con formato SAM o BAM [26], que contienen los alineamientos de las lecturas de entrada con el genoma de referencia. Ambos ficheros disponen de la misma información, con la diferencia que el primero de ellos es un fichero de texto, mientras que el segundo es un fichero binario comprimido. Además, en el caso de ARN, se genera otro fichero de texto (generalmente en formato BED [51]) con las coordenadas de los puntos de unión entre los exones que han sido identificados en el procesamiento.

```
seq.1a    0    1    21263612    50    40M1D10M    *    0    0
AGATGCTCACATAAAGCTCTGAGACTCCCAAATGGGTGACATGCTCAGTG
<=<=>===>==========<==>=>=>=======<=<=<>==<=>=<>=<
AS:255    NM:1    NH:1
```

Figura 2.2: Ejemplo de alineamiento de una lectura

En la Figura 2.2 se muestra el contenido de un fichero tipo SAM para una secuencia de ADN/ARN. Como puede verse, para cada alineamiento se almacena el identificador de la lectura en primer lugar (en nuestro ejemplo, «seq.1a»). A continuación, se almacena un valor decimal (en la Figura 2.2, «0») que contiene de forma codificada diferente información del alineamiento, como es la hebra en la que ha sido mapeada (positiva o negativa), si la lectura ha sido mapeada, si es el único alineamiento que se ha encontrado, etc. Siguiendo con la Figura 2.2, después se almacena el número de cromosoma (en la Figura 2.2, «1») y la posición genómica donde se ha encontrado la lectura en dicho cromosoma (en la Figura 2.2, «21263612»). El siguiente valor (en la Figura 2.2, «50»), es una puntuación de la calidad de alineamiento de la lectura. A continuación, se almacena el CIGAR, que contiene según la codificación fijada por la biblioteca SAMtools [26], una descripción detallada de cómo se alinea la lectura contra el genoma de referencia. El formato del CIGAR se basa en el siguiente patrón: primero se indica el número de nucleótidos implicados en una operación, y a continuación, un símbolo que codifica la operación que les afecta. En un CIGAR pueden haber varios números y códigos distintos consecutivos, indicando diferentes acciones sobre distintas partes de una lectura. Los principales códigos que se pueden encontrar en el CIGAR son:

M para indicar tanto aciertos como fallos de nucleótidos.

N para indicar que en dicha operación existe un punto de unión entre exones y, por lo tanto, los nucleótidos comprendidos por esta operación no aparecen en la lectura (es decir, se trata de un intrón). En el caso de tratarse de secuencias de ADN, esta operación no aparece ya que los intrones sí están presentes.

H/S para indicar un *Hard/Soft Clipping*, es decir, que ha quedado sin alinear un fragmento muy pequeño de la lectura localizado en cualquiera de los dos extremos.

I para indicar que en esa posición de la lectura existe una inserción, es decir, nucleótidos que aparecen en la lectura y no aparecen en el genoma de referencia.

D para indicar que se encuentra un borrado, es decir, hay nucleótidos en el genoma de referencia que no aparecen en la lectura.

Según la anterior descripción, el CIGAR mostrado en la Figura 2.2 «40M1D10M», indica que para esa lectura se han encontrado: 40 aciertos de nucleótidos, 1 borrado y, finalmente, otros 10 aciertos.

La información almacenada en el fichero mostrado en la Figura 2.2 continúa con una serie de valores que se utilizan cuando las secuencias de entrada no son *single-end*, sino *paired-end* o *mate-pair* [14]. La última información que se encuentra es la propia lectura mapeada junto con su calidad (codificada mediante una serie de códigos ASCII) y una serie de parámetros opcionales que son fijados por el alineador que ha generado el alineamiento.

Con los resultados de los ficheros SAM o BAM, se pueden obtener una serie de estadísticas sobre el mapeo. Este proceso se lleva a cabo mediante alguna herramienta especializada, como por ejemplo, RNA-SeQC [11]. Ésta permite obtener medidas de control de calidad de los alineamientos generados.

Finalmente, el estudio de ADN/ARN se completa con el análisis detallado de los resultados obtenidos [29]. Existen varios tipos de estudios que pueden realizarse y multitud de herramientas para ello. Por un lado, existe el estudio de variabilidad, que consiste en comparar múltiples genomas de personas sanas con otros de personas con alguna determinada enfermedad. De esta forma, se puede analizar cómo afecta dicha enfermedad al genoma humano. Otro posible análisis, es el estudio del efecto de un determinado fármaco sobre un individuo enfermo. Esto se realiza secuenciando el genoma de un individuo con una determinada enfermedad antes de aplicarle el medicamento. Una vez suministrado el fármaco y transcurrido un cierto tiempo, se vuelve a secuenciar al individuo para comprobar cómo ha afectado el tratamiento genéticamente a éste y a la enfermedad. Existen una gran variedad de aplicaciones para llevar a cabo dichos estudios, tales como Cufflinks, DESeq2, limma, etc.

2.2 Técnicas y alineadores actuales de ADN/ARN

Existe una gran variedad de técnicas para el alineamiento de secuencias de ADN/ARN. Entre todas estas técnicas, las más utilizadas son las basadas en: la transformada de Burrows-Wheeler (BWT) [5], el índice de texto completo en un espacio reducido (FMindex) [13], la colección de sufijos (SA) [30] y el algoritmo de Smith-Waterman (SWA) [46].

BWT es una técnica que consiste en la transformación de una cadena de texto en otra diferente. Esta nueva cadena contiene los mismos caracteres que la original pero en un orden distinto y repetidos varias veces seguidas. Esta característica es fácilmente aprovechable por un algoritmo compresor. Para el caso que nos ocupa, esto nos permite trabajar con el genoma de referencia comprimido y utilizar algoritmos como Bowtie o Bowtie 2 para buscar subcadenas de texto dentro de la cadena transformada (en nuestro caso, el genoma de referencia). Finalmente, cabe destacar, que la ventaja principal de este método es que tiene menos requisitos de memoria RAM que el resto.

FM-index es un índice de texto completo comprimido basado en BWT. Éste puede usarse para encontrar de forma eficiente las ocurrencias de un patrón dentro del texto comprimido, además de poder encontrar las posiciones que ocupan cada una de estas cadenas.

SA se basa en una colección ordenada de cadenas de texto denominada índice. Éste contiene todos los posibles sufijos de la cadena de texto original ordenados alfabéticamente. Para el alineamiento de secuencias, se realiza una búsqueda binaria sobre este índice. Además, esta búsqueda se puede optimizar mediante la utilización de tablas *hash*. Esta técnica conlleva unos requisitos de memoria considerablemente más elevados que BWT, pero a cambio, es generalmente más rápida.

Finalmente, el algoritmo de Smith-Waterman (SWA) cuantifica la similitud entre dos cadenas de texto comparando segmentos (i.e., subcadenas) de todos los posibles tamaños. Esta técnica es la más sensible de todas las nombradas hasta el momento, pero su coste computacional es muy elevado. Esto hace imposible utilizarla para búsquedas en genomas muy grandes, como es el caso del ser humano.

Todas estas técnicas descritas son usadas por la mayoría de los alineadores de lecturas de ADN/ARN que existen. De entre todos estos alineadores, algunos de los más destacables se recogen en el Cuadro 2.1.

Como se puede ver en el Cuadro 2.1, existe una gran cantidad de alineadores para el procesamiento de secuencias de ADN. Este cuadro indica junto a cada alineador el tipo de implementación HPC (*High Performance Computing*) que ofrece. Como se puede ver, la gran mayoría de alineadores proporciona únicamente soporte para sistemas multiprocesador.

La técnica basada en BWT es la más usada entre los mapeadores de secuencias de ADN con soporte multiprocesador. Esta técnica procesa de forma eficiente las lecturas con pocos errores. Por el contrario, cuando las lecturas contienen muchos errores o inserciones/borrados de muchos nucleótidos se comporta de forma ineficiente. A pesar de esto, alineadores como Bowtie, Bowtie 2 [23], BWA, BLASR [6] y BWA-MEM [25] son capaces de mapear un gran número de lecturas en un tiempo razonable y con una gran sensibilidad. Cabe destacar, que dentro de este grupo, el alineador BWA-MEM ofrece un gran rendimiento cuando el tamaño de las lecturas aumenta.

Otras aplicaciones como SOAP 3 [28] o Isaac [44] también procesan las secuencias de ADN de forma eficiente, aunque su rendimiento decae cuando el número de mutaciones aumenta.

Por otro lado, alineadores como Cushaw2-GPU y Soap3-dp obtiene una mayor capacidad de cómputo mediante el paralelismo híbrido entre GPU y CPU. En concreto, Cushaw2-GPU usa una implementación del algoritmo SWA desarrollada mediante la biblioteca CUDA para GPUs. Por el contrario, Soap3-dp usa una versión dinámica de la técnica BWT para el alineamiento de las lecturas en la GPU.

Finalmente, aplicaciones como Halvade ofrecen una implementación clúster basada en MapReduce 2.0. Para llevar a cabo la paralelización, el fichero de entrada se divide en paquetes que se procesarán de forma individual entre los nodos del clúster. Finalmente, los resultados generados por cada uno de los nodos del clúster se mezclarán en un único fichero de salida.

Por otro lado, los alineadores más destacados para el alineamiento de secuencias de ARN son Tophat, MapSplice y STAR. Todos ellos explotan el paralelismo a nivel de multiprocesador. Además, el único alineador de ARN que ofrece soporte para clústeres es RUM.

Tophat [50] es una de las aplicaciones más utilizadas para llevar a cabo el alineamiento de secuencias de ARN. Esta aplicación obtiene unas prestaciones muy buenas, en términos de sensibilidad, para lecturas de pequeña longitud. En cambio, en cuanto el tamaño de las lecturas aumenta, la sensibilidad disminuye progresivamente. Para llevar a cabo el alineamiento de las lecturas utiliza en primer lugar Bowtie. A continuación, crea una serie de genomas de referencia nuevos a partir de la información obtenida durante procesamiento anterior. Finalmente, las lecturas no mapeadas, vuelven a ser procesadas utilizando los nuevos genomas de referencia.

MapSplice [54], al igual que TopHat, utiliza Bowtie para realizar el mapeo de las lecturas. Su procesamiento se divide en dos fases. La primera realiza fragmentos de las lecturas para poder encontrar alineamientos exónicos. Cuando los fragmentos no son mapeados, se considera que contienen un punto de unión, y ayudándose de los fragmentos adyacentes mapeados, intenta encontrarlo. En la segunda fase del procesamiento, se genera una puntuación para los puntos de unión encontrados, y de esta manera poder filtrar y descartar falsos positivos.

Por último, STAR [12] es otro alineador que ofrece grandes prestaciones, tanto desde el punto de vista de sensibilidad como de rendimiento. Esto se debe a que la técnica utilizada

	Alineadores (última versión)	Plataforma	Código Abierto
Alineadores ADN	Bfast (2011)	Multiprocesador	Si
	Blat (2012)	-	Si
	Bowtie (2015)	Multiprocesador	Si
	Bowtie2 (2015)	Multiprocesador	Si
	BWA (2014)	Multiprocesador	Si
	BWA-MEM (2014)	Multiprocesador	Si
	Cushaw2-GPU (2013)	Multiprocesador/GPU	Si
	Cushaw3 (2014)	Multiprocesador	Si
	ContextMap2 (2015)	Multiprocesador	Si
	GEM (2013)	Multiprocesador	No
	Gnumap (2011)	Multiprocesador	Si
	Halvade (2015)	Multiprocesador/Clúster	Si
	HPG Aligner DNA SA (2015)	Multiprocesador	Si
	Soap (2007)	Multiprocesador	Si
	Soap2 (2011)	Multiprocesador	No
	Soap3-dp (2011)	Multiprocesador/GPU	Si
Alineadores ARN	HPG Aligner RNA BWT (2014)	Multiprocesador	Si
	HPG Aligner RNA SA (2015)	Multiprocesador	Si
	MapSplice (2011)	Multiprocesador	Si
	Cluster HPG Aligner RNA BWT (2015)	Multiprocesador/Clúster	Si
	Olego (2015)	Multiprocesador	Si
	RNASEQR (2012)	Multiprocesador	Si
	RUM (2015)	Multiprocesador/Clúster	Si
	SpliceMap (2010)	Multiprocesador	Si
	STAR (2015)	Multiprocesador	Si
	TopHat (2012)	Multiprocesador	Si
	TopHat2 (2015)	Multiprocesador	Si

Cuadro 2.1: Breve listado de alineadores para ADN/ARN

para mapear lecturas en el genoma de referencia está basada en SA. El proceso de alineamiento también está formado por diversas etapas, como ocurría con Tophat. En primer lugar, mapea las lecturas sencillas que no contienen errores aplicando la técnica SA. En una segunda fase, las lecturas no mapeadas se fragmentan y se aplican técnicas de mapeo más sensibles para encontrar un alineamiento válido. La mayor velocidad de STAR frente a Tophat se debe, sobre todo, que trabaja con el genoma sin comprimir. El principal inconveniente de esta implementación es la cantidad de memoria necesaria para almacenar el índice (alrededor de 30 GiB).

CAPÍTULO 3

Métodos propuestos

En este capítulo se describen los métodos propuestos y las implementaciones realizadas en esta tesis para el alineamiento de secuencias de ADN/ARN, además, se detallan las técnicas HPC usadas en cada uno ellos. En cada uno de los alineadores desarrollados se describen las técnicas de alineamiento utilizadas y las técnicas HPC aplicadas para conseguir dotar a estos de las características requeridas: velocidad de procesamiento, sensibilidad en los resultados y gran capacidad de almacenamiento.

3.1 Introducción

El objetivo principal de los métodos propuestos en este capítulo es el de obtener mejores prestaciones que las de los alineadores actuales, tanto en la calidad de los mapeos encontrados como en el tiempo de procesamiento.

Los métodos propuestos utilizan para un primer alineamiento de las secuencias de entrada en el genoma de referencia un algoritmo basado, bien en la transformación de Burrows-Wheeler (BWT), o bien en la construcción de una matriz de sufijos (SA). A continuación, todos utilizan el algoritmo de Smith-Waterman para alinear las secuencias no encontradas previamente.

Todos los métodos desarrollados siguen un esquema común para el procesamiento de las lecturas. Inicialmente, el fichero de entrada se divide en paquetes que contienen un número fijo de lecturas. Estos paquetes atraviesan en orden las distintas etapas del procesamiento. Cada vez que una etapa finaliza el procesamiento de un paquete, este se almacena en una cola que une esa etapa con la siguiente. Cuando la siguiente etapa queda disponible, el paquete abandona la cola para continuar con su procesamiento. Cuando el paquete llega a la última etapa del cauce, los alineamientos encontrados se escriben en el fichero de salida.

La principal ventaja de este diseño segmentado es que si el sistema dispone de suficientes recursos, permite explotar tanto el paralelismo de grano grueso, a nivel de paquete, como el de grano fino, a nivel de las lecturas de un paquete. Además, esta forma de procesamiento reduce las necesidades de memoria, ya que el fichero de entrada se divide en paquetes de lecturas más pequeños.

Aprovechando lo anterior, se han realizado diferentes implementaciones HPC para el procesamiento paralelo de las lecturas. Por un lado, para sistemas multiprocesador, y por otro, para sistemas clúster con multitud de nodos.

Para diferenciar los distintos métodos desarrollados, se les ha dotado de un nombre representativo. Todos comienzan por «HPG Aligner» seguido de varios códigos que indican: si el alineamiento es de ADN o ARN, la técnica de mapeo utilizada y la paralelización aplicada. Los códigos concretos se mostrarán a medida que se presenten los distintos alineadores.

3.2 Método de alineamiento de lecturas de ARN basado en BWT

El primer método presentado para el procesamiento de secuencias de ARN se basa en la transformada de Burrows-Wheeler (BWT). Esta técnica se aplica para el alineamiento de las lecturas más sencillas, es decir, las lecturas que contienen como máximo un error. Para las lecturas con un mayor número de errores se utiliza el algoritmo SWA, ya que este ofrece una mayor sensibilidad.

La implementación HPC de este algoritmo se ha enfocado a sistemas multiprocesador. Una primera versión de la implementación HPC se ha desarrollado con hilos OpenMP y asignación estática de hilos a tareas. También se ha desarrollado una segunda versión con hilos POSIX y asignación dinámica de hilos a tareas.

3.2.1 Algoritmo

En este apartado se describe cada una de las etapas que forman parte del procesamiento del algoritmo denominado HPG Aligner ARN BWT [36].

El algoritmo está dividido en un total de 6 etapas secuenciales (A-F). La primera de estas etapas lee las secuencias de ARN almacenadas en uno o varios ficheros en formato Fasta o FastQ. Estas secuencias forman los paquetes que irán pasando por las distintas etapas del procesamiento. El resultado final generará un fichero en formato SAM/BAM con los alineamientos de las lecturas, además de un fichero en formato BED con las coordenadas de los puntos de unión.

A continuación se detallan cada una de las etapas de este algoritmo.

Etapa A (lectura de datos). Esta etapa transfiere las secuencias de ARN desde el fichero de entrada, almacenado en el disco duro, a la cola inicial del algoritmo (en la Figura 3.1, «Read Queue»). Por lo tanto, la etapa A «Read Data» de la Figura 3.1 formará paquetes de «n» lecturas y los almacenará en esta cola (el tamaño del paquete es configurable por el usuario, por defecto, está fijado a 200 lecturas). Para almacenar estos paquetes de lecturas, se utiliza una lista de estructuras de datos. Estas estructuras contienen toda la información que requieren las etapas posteriores.

Puesto que los ficheros de entrada se encuentran habitualmente comprimidos en formato gzip debido a su gran tamaño, el software desarrollado ofrece la posibilidad de leer este tipo de ficheros sin que sea necesario descomprimirlos previamente.

Etapa B (BWT, transformada de Burrows-Wheeler). La etapa B, «BWT», de la Figura 3.1, realiza un mapeo rápido de las lecturas no conflictivas usando una implementación propia de la transformada de Burrows-Wheeler [49]. Esta implementación permite hasta 1 EID (cambio de base, inserción o borrado) por lectura.

El resultado final de esta etapa son los alineamientos de las lecturas. Estos contienen el cromosoma donde se encuentra, la hebra, la posición genómica, etc. Toda esta información se almacena en el paquete junto a la lectura correspondiente.

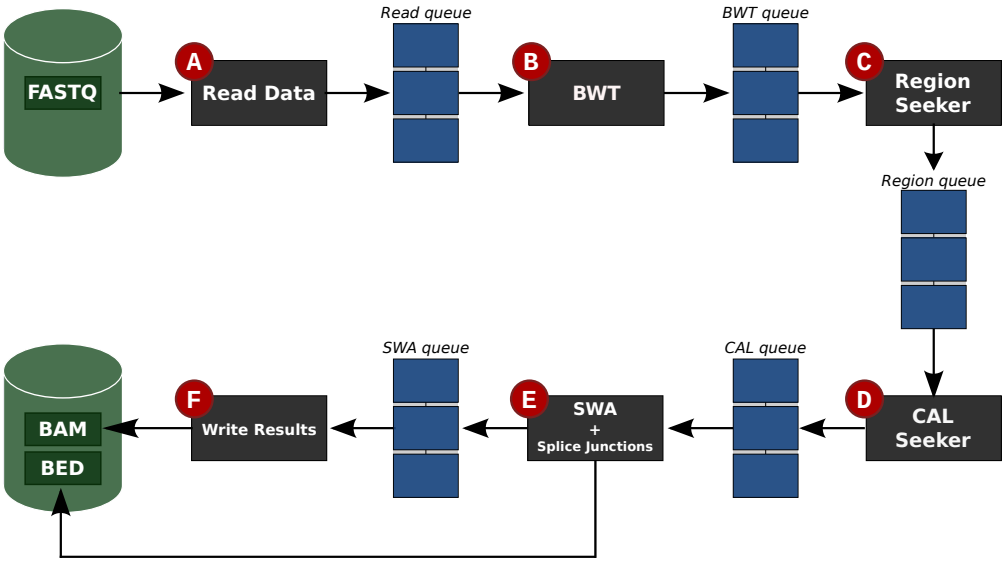


Figura 3.1: Diagrama de las etapas de HPG Aligner ARN BWT-S

A partir de esta etapa existen dos tipos de lecturas: las no mapeadas, que serán procesadas en la siguiente etapa, y las mapeadas. Estas últimas ya disponen de la información de mapeo, y por lo tanto, durante la última etapa de procesamiento, se almacenarán en el fichero de salida.

Etapa C (buscador de regiones). La etapa C, «Region Seeker», de la Figura 3.1, divide las lecturas no mapeadas en una serie de fragmentos o semillas superpuestas de tamaño 15 nts. Por ejemplo, para una lectura de 100 nt, se obtienen 8 semillas empezando desde el primer nucleótido, además de las 7 semillas solapadas, como puede observarse en la Figura 3.2. Esta fragmentación de la lectura original cuenta con que los errores de alineamiento se concentren en unas pocas semillas y que, por tanto, el resto de semillas puedan mapearse sin problemas.

A continuación, el algoritmo BWT se emplea de nuevo para mapear cada una de estas semillas en el genoma de referencia. A diferencia de la etapa anterior, en este alineamiento no se permiten EIDs, ya que esto incrementaría considerablemente el tiempo de procesamiento. Además, al tratarse de secuencias muy cortas, generaría muchos falsos positivos.

El resultado final de esta etapa es una colección de regiones. Estas regiones identifican la localización de cada una de las semillas mapeadas en el genoma de referencia, y se utilizarán en la siguiente etapa de procesamiento para formar las CALs.

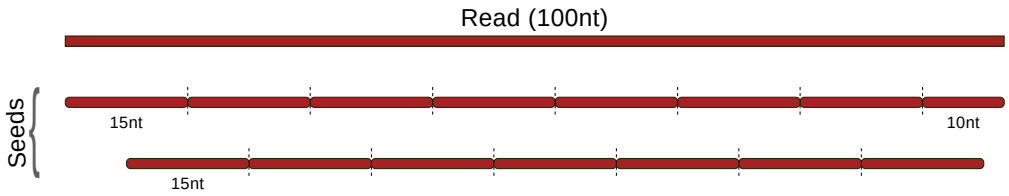


Figura 3.2: Ejemplo de fragmentación (semillas) de una lectura de 100 nt

Etapa D (buscador de CALs). La etapa D, «CAL Seeker», de la Figura 3.1, une las regiones asociadas a una lectura cuya distancia genómica es menor a un número determinado de nucleótidos. Este paso es necesario debido a que las semillas son muy pequeñas y el número de mapeos encontrados en el genoma a partir de ellas es muy elevado. Por lo tanto, solo se considerarán zonas potenciales de mapeo, aquellas que contengan un número determinado de semillas. Una región fusionada se promueve a una localización candidata de alineamiento (CAL) si su longitud es mayor a un umbral dado. Por lo tanto, las CALs son áreas donde una lectura tiene una probabilidad muy alta de mapeo.

En este caso, las lecturas que se están procesando provienen de muestras de ARN. Esto implica, que a la hora de procesar las CALs de una determinada lectura, hay que tener en cuenta que ésta puede contener puntos de uniones entre exones.

En la Figura 3.3, pueden observarse dos CALs, «CAL_0» y «CAL_1» formadas a partir de una lectura no mapeada en la etapa B. Además, «CAL_0» contiene dos semillas solapantes que no se han conseguido mapear debido a algún o algunos tipos de EIDs. Por otro lado, «CAL_1» cubre otra parte de la lectura que corresponde a otro exón y se ha formado mediante otro conjunto de semillas mapeadas.

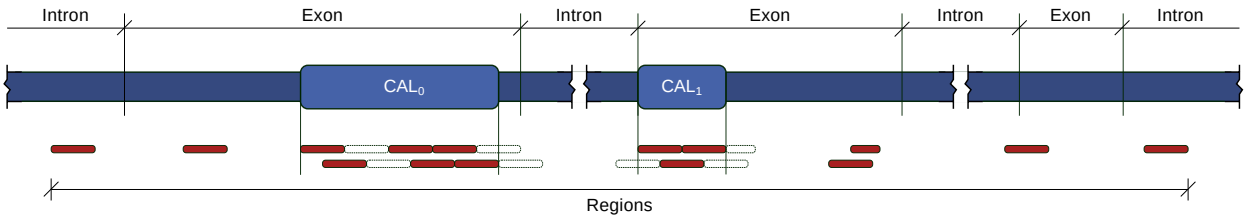


Figura 3.3: Identificación de CALs a partir de regiones

Etapa E (SWA + buscador de uniones entre exones). La etapa E, «SWA + Splice Junctions», de la Figura 3.1, busca el alineamiento de las lecturas más difíciles y los puntos de unión entre exones. Para ello, se llevan a cabo las siguientes operaciones:

(i) En primer lugar, las CALs de una lectura se extienden por ambos extremos un número fijo de nucleótidos (por defecto, 30 nts). Esto puede incluir fragmentos de intrones dentro de las CALs. Este proceso puede observarse gráficamente en la parte superior de la Figura 3.4.

(ii) A continuación, las CAL separadas por una distancia inferior al tamaño máximo de intrón (500.000 nts) se fusionan, como puede observarse en la Figura 3.5.

(iii) En el siguiente paso se alinea la lectura frente al fragmento del genoma de referencia comprendido por las coordenadas de la/las CALs. Para esto, se utiliza la versión optimizada del algoritmo SWA [46]. Este explota las instrucciones SSE, permitiendo empaquetar los datos de 4 en 4 y realizar 4 alineamientos en paralelo. Finalmente, se obtendrá el alineamiento entre la lectura y cada CAL extendida y su calidad.

(iv) Para los alineamientos encontrados en los pasos anteriores, y que superen una calidad de alineamiento superior a un umbral dado, se genera el registro de alineamiento correspondiente. Por otro lado, se buscan los puntos de unión entre exones. Para esto, se buscan las lecturas cuyos alineamientos contengan un gran número de borrados consecutivos. Esto se asume a que es debido que al extender las CALs, se habrá cogido parte

de un intrón. Una vez detectado este efecto, se realiza la búsqueda de las coordenadas exactas del inicio y fin del intrón. Estas coordenadas son identificables mediante las marcas «GT-AG» y «CT-AC». Todas las uniones de exones y el número de veces que cada una de ellas se ha detectado, se almacenan en una estructura basada en un árbol binario de búsqueda (AVL). Se ha utilizado esta estructura de datos debido a que esta información se irá actualizando durante todo el proceso y permite optimizar tanto el espacio utilizado en memoria, como la velocidad de acceso a información.

(v) La última tarea de esta etapa consiste en escribir la información almacenada en el AVL en un fichero con formato BED [51]. Cabe destacar, que esta información solo se escribirá en el disco duro una vez el último paquete de lecturas haya sido procesado.

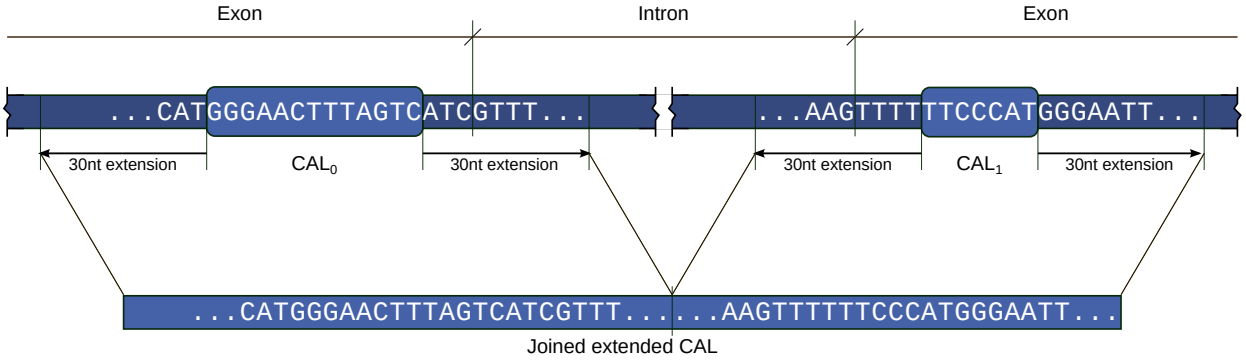


Figura 3.4: Extensión de CALs por ambos extremos para fusionarlas posteriormente

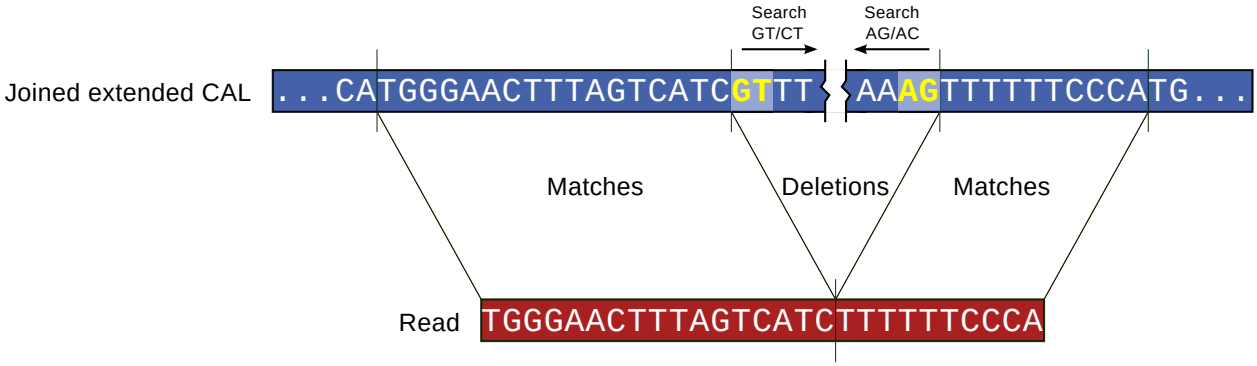


Figura 3.5: Búsqueda de los indicadores del punto de unión entre exones

Etapa F (escritura de resultados). La etapa F «Write Results» de la Figura 3.1, completa el procesamiento de un paquete de lecturas. Esto implica escribir el resultado de los alineamientos en el fichero de salida en el disco duro. Para su almacenamiento existen dos posibilidades: el formato SAM (los datos se escriben en texto plano) o el formato BAM (se aplica una compresión) [26]. Cada uno de estos formatos tiene sus ventajas e inconvenientes. Si se almacenan los alineamientos en formato BAM, el espacio en el disco duro se ve reducido de forma drástica, en cambio, se incrementa de forma considerable el tiempo de procesamiento. Por otro lado, el formato SAM implica ahorrarse el tiempo de compresión, pero el espacio de

almacenamiento en el disco duro es mucho más elevado. El usuario es quien selecciona qué tipo de formato de salida prefiere (el formato por defecto es BAM).

3.2.2 Paralelización basada en la asignación estática de tareas a hilos

En este apartado se describe con detalle la paralelización de la implementación anterior para un sistema multiprocesador. Esta implementación se denomina HPG Aligner ARN BWT-S. La organización de este alineador contempla dos tipos de concurrencia paralela, ambas enfocadas a sistemas multiprocesador. Por un lado, se aplica el paralelismo a nivel de paquete donde cada hilo procesa un paquete de lecturas distinto. Por otro lado, si existe un número suficiente de procesadores en el sistema, se aplica el paralelismo intrínseco a determinadas etapas, donde varios hilos procesan una misma etapa y, por lo tanto, el paralelismo se aplica a nivel de lecturas.

Esta forma de funcionamiento simula un procesador superescalar por etapas, tal y como se muestra en las Figuras 3.6a y 3.6b. En la primera de ellas, puede observarse la traza de una ejecución secuencial, donde todas las etapas se ejecutan con un solo hilo. En la traza puede observarse como cada etapa tiene que esperar a que finalice la etapa anterior para poder continuar con el procesamiento. La Figura 3.6b, por contra, muestra el caso en el que varios hilos intervienen en la ejecución, lo que permite solapar el procesamiento entre ellos. El propósito de explotar este último paralelismo es doble, por un lado acelerar el procesamiento, y por otro, equilibrar las velocidades de las distintas etapas. Si no se realizara este último paralelismo, el rendimiento del algoritmo completo vendría determinado por el tiempo de ejecución de la etapa más lenta.

Esta implementación de HPG Aligner ARN BWT-S se ha llevado a cabo utilizando la biblioteca OpenMP. Esta biblioteca facilita la paralelización de códigos secuenciales. Por ejemplo, utilizando la directiva «omp parallel sections» para marcar qué tareas son independientes y se pueden paralelizar ; o utilizando la directiva «omp parallel for» para indicar que las iteraciones de un bucle pueden repartirse entre varias hebras. Como se puede ver en el Listado 3.1, el primer nivel de paralelización ha consistido en marcar cada una de las etapas del algoritmo como una sección que puede ejecutarse de forma independiente en OpenMP. A cada una de estas secciones se les ha asignado un número de hilos que se calcula en función del coste relativo de la etapa que engloban y del número total de hilos disponibles. Además, el código de las distintas etapas también se ha paralelizado, como se indica a continuación.

```
#define num_stages      6
#define AB_pending_work TRUE
#define BC_pending_work TRUE
#define CE_pending_work TRUE
#define EF_pending_work TRUE

#pragma omp parallel sections num_threads( num_stages ) {
  #pragma omp section {
    A_Read_Data( FASTQ_file, Read_queue );            // Read batches from FASTQ file
  }
  #pragma omp section {
    B_BWT( Read_queue, BWT_queue );                   // Apply the BWT
  }
  #pragma omp section {
    C_Region_Seeker( BWT_queue, Region_queue );       // Apply Region seeker
  }
  #pragma omp section {
    D_CAL_Seeker(  Region_queue, CAL_queue );         // Apply CAL seeker
  }
  #pragma omp section {
    E_SWA_Splice_Junctions(   CAL_queue, SWA_queue ); // Apply SWA and detect splice jcts
    E_Write_Splice_Junctions( BED_file );             // Writes results onto BAM file
  }
  #pragma omp section {
    F_Write_Data( SWA_queue,  BAM_file );             // Writes results onto BAM file
  }
}
```

Listado 3.1: Versión simplificada del algoritmo paralelizado medinate OpenMP

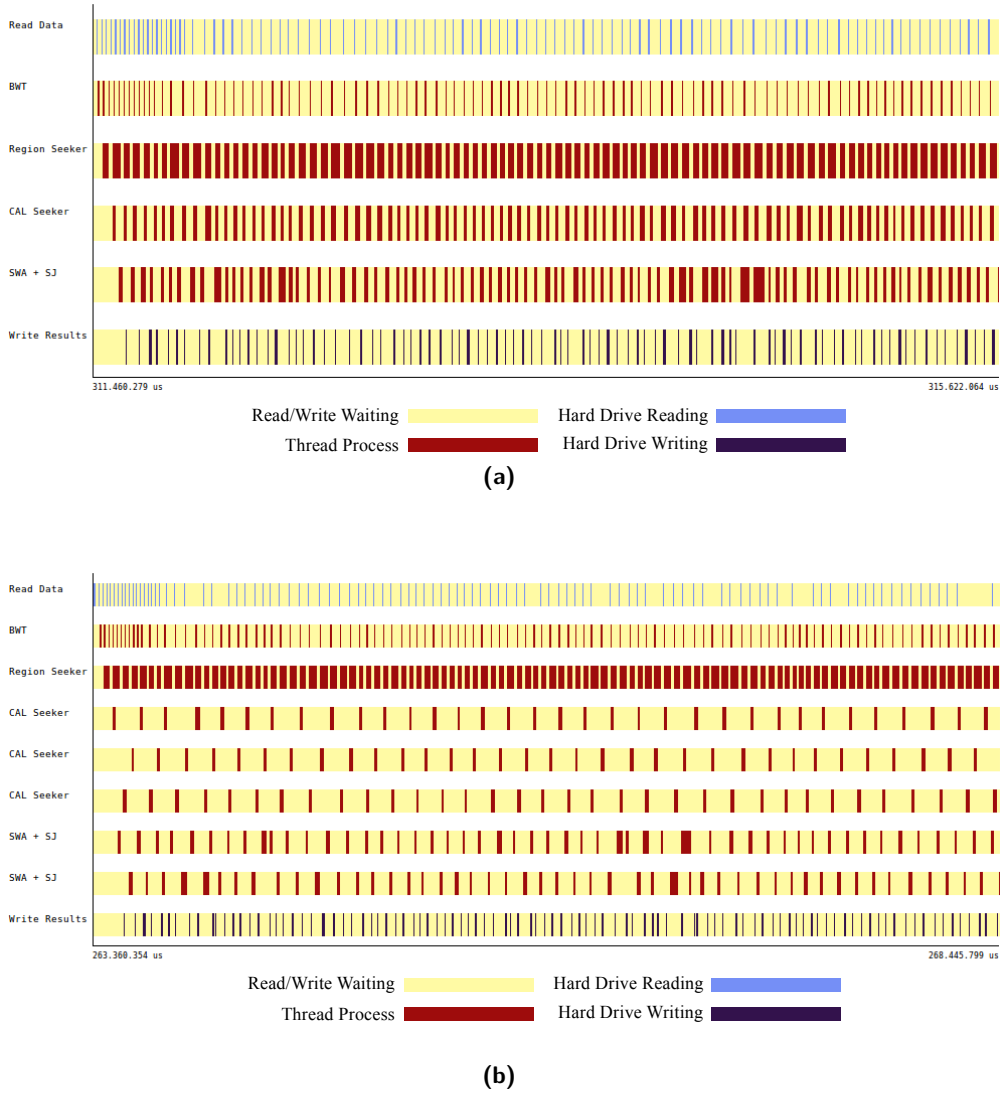


Figura 3.6: Ejecución del flujo del alineador. (a) Etapas secuenciales. (b) Explotación paralelismo etapas

Etapas A y F. Estas etapas son las encargadas de transferir los datos de entrada y salida entre el disco duro y el alineador. Además, ambas etapas se ejecutan de forma secuencial, es decir, leen o escriben los paquetes de uno en uno. El Listado 3.2 muestra un resumen de las operaciones realizadas en ambas etapas. En este listado, puede observarse como la etapa A lee los datos formando paquetes a partir del fichero «FASTQ_File» para almacenarlos en la cola intermedia «Read_queue». Una vez se detecta el final del fichero, se notifica este hecho a la siguiente fase fijando la variable «AB_pending_work» a *FALSE*.

La etapa F realiza el proceso inverso, es decir, lee los alineamientos encontrados de la cola «SWA_queue» y escribe la información en el fichero de salida en formato BAM/SAM. Estas tareas las realizan las funciones «F_Read_Batch» y «F_Write_Batch», respectivamente.

Las etapas A y F son de E/S, por lo tanto, solo se les asigna un hilo a cada una de ellas. Esto es debido a que nuestro objetivo es implementar una aplicación que pueda ejecutarse en servidores de sobremesa. Por lo tanto, asociar más de un hilo a estas dos etapas no supondría ningún tipo de ganancia.

```
void A_Read_Data( FASTQ_file_t FASTQ_file, Read_queue_t Read_queue ) {
  while ( !end_file( FASTQ_file ) ) {
    A_Read_Batch(  FASTQ_file,   Batch ); // Read a single batch from
    A_Write_Batch( Read_queue,   Batch ); // FASTQ file onto Read_queue
  }
  AB_pending_work = FALSE;
}

void F_Write_Data( SWA_queue_t SWA_queue,  BAM_BED_file_t BAM_BED_file ) {
  while ( ( EF_pending_work ) OR ( !empty_queue( SWA_queue ) ) ) {
    F_Read_Batch(  SWA_queue,    Batch ); // Write a single batch from SWA_queue
    F_Write_Batch( BAM_file,     Batch ); // onto BAM file
  }
}
```

Listado 3.2: Etapas *A* y *F*. Lectura y Escritura de Datos

Etapas B y C. La etapa B se encarga de aplicar el algoritmo BWT a cada una de las lecturas del paquete. La paralelización de esta etapa consiste en ejecutar concurrentemente esta técnica sobre diferentes lecturas, es decir, paralelizar el bucle *for* de la función «B_BWT» que se muestra en el Listado 3.3. Este tipo de paralelización se puede llevar a cabo sin problema ya que no existe ningún tipo de dependencia entre las lecturas del paquete.

El número de hilos que van a estar colaborando para procesar esta etapa se fija en la variable «nt_BWT». El procesamiento de esta etapa finaliza cuando la variable «AB_pending_work» tiene el valor *FALSE* y la cola «Read_queue» está vacía. Cuando se produce esta situación, se fija el valor de la variable «BC_pending_work» a *FALSE*. Es importante resaltar que en esta etapa, dado el elevado número de lecturas por paquete y el moderado número de procesadores de los servidores actuales, la concurrencia obtenida en el procesamiento de un paquete es elevada.

El funcionamiento de la etapa C sigue el mismo mecanismo que el de la etapa B. La única diferencia, es que la concurrencia de paralelismo en esta fase es menor. Esto se debe a que en esta fase solo llegan las lecturas no mapeadas por la etapa anterior.

```
#define nt_BWT           W // User-defined parameter
#define nt_Region_Seeker X // User-defined parameter

void B_BWT( Read_queue_t Read_queue, BWT_queue_t BWT_queue ){
  while ( ( AB_pending_work ) OR ( !empty_queue( Read_queue ) ) ) {
    B_BWT_Read(  Read_queue, Batch );
    #pragma omp parallel for num_threads( nt_BWT ) {
      for ( i = 0; i < Batch.n_reads; i++ )   // Loops over all reads of the batch
        B_BWT_Single_Read( Batch.read[ i ] ); // Applies the BWT to the i-th read
    }
    B_BWT_Write( BWT_queue, Batch );
  }
  BC_pending_work = FALSE;
}

void C_Region_Seeker( BWT_queue_t BWT_queue, Region_queue_t Region_queue ){
  while ( ( BC_pending_work ) OR ( !empty_queue( BWT_queue ) ) ) {
    C_Region_Read(  BWT_queue,    Batch );
    #pragma omp parallel for num_threads( nt_Region_Seeker ) {
      for ( i = 0; i < Batch.n_reads; i++ )      // Loops over all reads of the batch
        C_Region_Single_Read( Batch.read[ i ] ); // Applies Region seeker to the i-th read
    }
    C_Region_Write( Region_queue, Batch );
  }
  CD_pending_work = FALSE;
}
```

Listado 3.3: Etapas B y C. Procesamiento de las lecturas dentro de un paquete en paralelo

Etapas D y E. Estas dos etapas se encargan de buscar las posibles zonas genómicas donde pueden mapearse las lecturas más conflictivas. Además, la etapa E utiliza SWA para buscar el alineamiento de estas lecturas y los puntos de unión entre exones en caso de existir. La paralelización aplicada en ambas etapas se realiza a nivel de paquete, es decir, en el procesamiento de varios paquetes. Se ha optado por este tipo de paralelización debido a que estas etapas solo llegan las lecturas no mapeadas, y por lo tanto, el número de lecturas es muy reducido con respecto al inicial.

Las variables «nt_CAL_Seeker» y «nt_SWA_Splice_Junction» establecen el número de hilos que participan en las etapas C y D, respectivamente.

```
#define nt_CAL_Seeker           Y // User-defined parameter
#define nt_SWA_Splice_Junctions Z // User-defined parameter

void D_CAL_Seeker( Region_queue_t Region_queue, CAL_queue_t CAL_queue ){
  #pragma omp parallel private( Batch ) num_threads( nt_CAL_Seeker ) {
    while ( ( CD_pending_work ) OR ( !empty_queue( Region_queue ) ) ) {
      D_CAL_Seeker_Read(  Region_queue, Batch );
      D_CAL_Seeker(                     Batch ); // Applies CAL seeker to all reads
      D_CAL_Seeker_Write( CAL_queue,    Batch ); // within the batch
     }
  }
  DE_pending_work = FALSE;
}

void E_SWA_Splice_Junctions( CAL_queue_t CAL_queue, SWA_queue_t SWA_queue ){
  #pragma omp parallel private( Batch ) num_threads( nt_SWA_Splice_Junctions ) {
    while ( ( DE_pending_work ) OR ( !empty_queue( CAL_queue ) ) ) {
      E_SWA_Splice_Junctions_Read(  CAL_queue, Batch );
      E_SWA_Splice_Junctions(                  Batch ); // Applies SWA/detecs splice jcts
      E_SWA_Splice_Junctions_Write( SWA_queue, Batch ); // to/in all reads within the batch
    }
  }
  EF_pending_work = FALSE;
}
```

Listado 3.4: Etapas D y E. Procesamiento de paquetes de lecturas en paralelo

Debido a las limitaciones de la biblioteca OpenMP 3.0 a la hora de gestionar tareas y las dificultades en la implementación del algoritmo, ha sido necesario adoptar diversas decisiones de diseño (y, en consecuencia, de rendimiento). En concreto, debido a la explotación del paralelismo anidado de OpenMP, el número de hilos que está en funcionamiento en cada etapa de HPG Aligner ARN BWT-S tiene que ser fijado en tiempo de compilación. Por este motivo, cuando un hilo queda bloqueado por la falta de trabajo, el núcleo del procesador asociado a ese hilo queda ocioso. Esto conlleva el desperdicio de los recursos del sistema. Por otro lado, cuando el número de hilos es bajo o moderado, esta asignación estática de hilos a etapas limita el rendimiento del proceso. Esto se debe, a que como puede observarse en la Figura 3.6b, la etapa C es muy costosa y, por lo tanto, hasta que no finalice el procesamiento de sus paquetes, las etapas posteriores quedarán bloqueadas. Esto convierte a la etapa C en un cuello de botella.

3.2.3 Paralelización basada en la asignación dinámica de tareas a hilos

Como se ha visto en el apartado anterior, la asignación estática de tareas presenta el problema de que la etapa más costosa se convierte en un cuello de botella e impide aprovechar al máximo la cantidad de procesadores disponibles. Una forma de solucionar esta limitación es la de asociar los hilos a las distintas etapas de forma dinámica. Para ello, en lugar de utilizar OpenMP, se ha realizado una nueva implementación basada en hilos POSIX. Mediante esta nueva forma de asignación de hilos a etapas se consigue que no haya hilos ociosos por culpa de que no haya trabajo para determinadas etapas.

Esta nueva organización también permite la aplicación de diversas técnicas de optimización. Por ejemplo, se puede priorizar la ejecución de determinados tipos de tareas, como por

ejemplo, las más costosas. De esta manera, se consigue reducir los tiempos de inactividad de hilos debido al desequilibrio de la carga de trabajo. Por otro lado, se puede redistribuir la carga de trabajo para favorecer la localidad de datos haciendo que un hilo realice el procesamiento completo de un paquete de lecturas. Finalmente, se puede aplicar la técnica de robo de trabajo para que los hilos ociosos «roben» tareas que inicialmente fueron asignadas a otros hilos y que ahora estén sobrecargados.

Este nuevo método que denominamos HPG Aligner ARN BWT-D, se diferencia del método anterior en que dispone de una cola de trabajo que almacena los paquetes de lecturas. Estos paquetes contienen una etiqueta que indica en qué etapa de procesamiento se encuentra. Por otro lado, tiene un conjunto de hilos dispuesto a ejecutar cualquier tipo de tarea. Estos hilos cogen los paquetes de esta cola de trabajo y llevan a cabo la tarea indicada en el paquete. Cuando finaliza el procesamiento, el hilo vuelve a dejar el paquete en la cola de trabajo y actualiza la etiqueta, quedando disponible para realizar una nueva tarea. Inicialmente hay un hilo encargado de formar los paquetes de lecturas e introducirlos en la cola de trabajo. Cuando la cola se llena, el hilo encargado de realizar la lectura del fichero de entrada pasará a procesar paquetes al igual que el resto. Cuando la cola vuelve a tener espacio para nuevos paquetes, el primer hilo que finaliza el procesamiento pasa a ser el nuevo lector y a crear paquetes desde el fichero. A la vez, otro hilo será el encargado de ir leyendo los datos generados por el procesamiento y de ir almacenándolos en el disco duro. Este hilo se comporta de forma similar al hilo lector: mientras hayan datos pendientes de ser almacenados, los irá almacenando; cuando no hayan datos, el hilo dejará de ser escritor para procesar los paquetes pendientes. Además, como se ha comentado previamente, las etapas pueden priorizarse en función de su coste de procesamiento. De esta forma, los hilos procesarán antes los paquetes que se encuentren en las etapas marcadas como más prioritarias. Para el caso de ARN, se ha priorizado en primer lugar la etapa SWA, seguida de la búsqueda de CALs y, para finalizar, la etapa BWT.

Los métodos de alineamiento de ARN que se describen a continuación extienden este método de asignación dinámica de tareas a hilos.

3.3 Implementación mejorada mediante la estructura metaexón

En este apartado se describe una nueva implementación de HPG Aligner ARN basada en el método anterior. En esta nueva versión se busca mejorar la sensibilidad de los resultados aumentando la velocidad de procesamiento. En concreto, el objetivo principal se centra en el refinamiento de las lecturas cuyos alineamientos contienen uno o varios *hard/soft clipping*. Este tipo de alineamientos ocurren cuando un extremo muy pequeño de la lectura no se consigue mapear. Esto puede ocurrir por dos razones: porque el fragmento no mapeado contenga uno o varios EID o porque pertenezca a otro exón.

En los métodos anteriores era imposible mapear correctamente estos pequeños fragmentos de lecturas. Esto se debía a que la longitud de las semillas era mayor que la del fragmento a mapear, y por lo tanto, era imposible localizarlo. Por otro lado, si se hubiera reducido el tamaño de semilla, además de incrementarse considerablemente el tiempo de procesamiento, se hubiera elevado demasiado el número de mapeos obtenidos, haciendo imposible la localización del lugar correcto.

3.3.1 Estructura metaexón

En este método, denominado HPG Aligner ARN BWT+M, se ha diseñado y utilizado una nueva estructura de datos denominada metaexón. En esta estructura se almacena toda la información de los alineamientos que se van obteniendo a medida que avanza el procesamiento. El

objetivo de esta estructura es el de poder alinear completamente las lecturas que con los métodos anteriores no podían alinearse, además de acelerar la velocidad de procesamiento.

Como puede observarse en la Figura 3.7, la estructura metaexón consiste en un vector de listas doblemente enlazadas. Cada posición del vector representa un cromosoma distinto. Por lo tanto, los exones que se encuentran durante el procesamiento se almacenarán en la lista doblemente enlazada asociada al cromosoma en el que se hayan encontrado. Por otro lado, cada exón contiene un listado con los inicios y finales de sus puntos de unión. Para esto, se almacena la referencia de cada uno de los nodos del árbol AVL, ya que este contiene todos los puntos de unión encontrados.

Mediante este mecanismo se consiguen alinear muchas de las lecturas conflictivas sin necesidad de tener que recurrir al algoritmo SWA. De esta manera, se acelerará el procesamiento, además de incrementar la sensibilidad y la especificidad de los resultados.

Esta estructura se comparte por todos los hilos durante el procesamiento. Por lo tanto, es necesario utilizar un mecanismo de sincronización que garantice el acceso exclusivo de solo uno de los hilos cuando sea necesario escribir en ella.

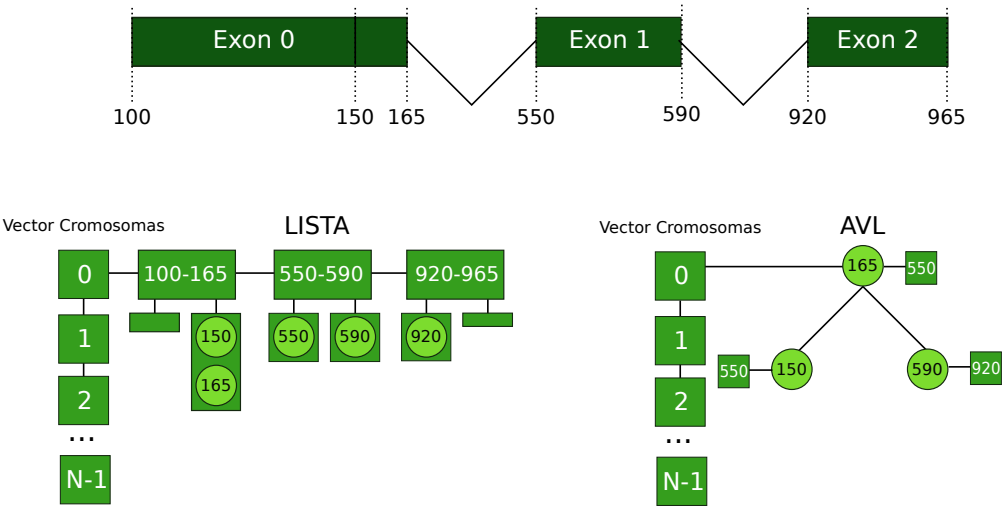


Figura 3.7: Esquema de la estructura metaexón

3.3.2 Algoritmo

Además de la estructura metaexón, el procesamiento completo se ha dividido en tres flujos de trabajo distintos. De esta manera, las lecturas más conflictivas pasarán a través de estos flujos de trabajo para intentar encontrar algún alineamiento correcto. Para lograr este objetivo, las etapas de los diferentes flujos de trabajo se ayudarán de la estructura metaexón, cuya información se irá actualizando durante todo el procesamiento.

La Figura 3.8 muestra como se han estructurado los diferentes flujos de trabajo. Además, como puede observarse, cada uno de estos flujos contiene una serie de etapas de procesamiento siguiendo el esquema de HPG Aligner ARN BWT-D.

Para llevar a cabo el procesamiento del conjunto de datos, inicialmente, el primer flujo procesa todas las lecturas. Durante este proceso, se almacena la información generada por los alineamientos en la estructura metaexón y en la AVL. Además, este flujo de trabajo generará tres tipos de salidas distintas que se almacenan en diferentes ficheros según se muestra en la Figura 3.8:

(i) Las lecturas alineadas por completo se almacenan en el fichero de salida etiquetado como «BAM» en la Figura 3.8.

(ii) Las lecturas que se han alineado más de la mitad, sin llegar a completar el alineamiento en la etapa BWT, se almacenan en el fichero etiquetado como «SINGLE ANCHOR» en la Figura 3.8.

(iii) Las lecturas que les falta alinear algún extremo muy pequeño en la etapa SWA se almacenan en el fichero temporal «HARD CLIPPING» de la Figura 3.8.

El segundo flujo de trabajo, lee las lecturas almacenadas en el fichero «SINGLE ANCHOR» e intenta completar su alineamiento. Para esto, utiliza la información contenida en las estructuras metaexón y AVL. Este flujo de trabajo genera dos tipos de salidas: las lecturas mapeadas correctamente y las lecturas mapeadas parcialmente que se almacenarán en el fichero temporal «HARD CLIPPING». Durante el procesamiento de este flujo de trabajo se actualizan de nuevo los datos de las estructuras metaexón y AVL.

Finalmente, el tercer y último flujo de trabajo lee las lecturas contenidas en el fichero «HARD CLIPPING». Este flujo de trabajo intenta refinar los alineamientos de las lecturas más conflictivas con la ayuda de la estructura metaexón.

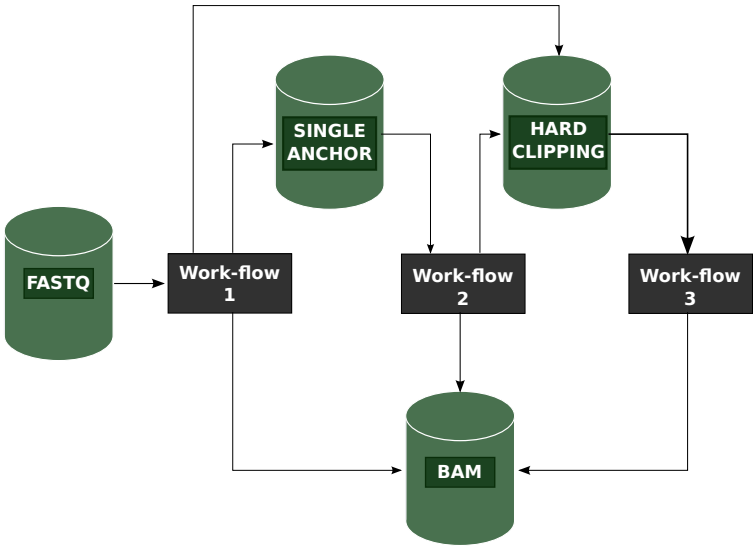


Figura 3.8: Esquema de los flujos de trabajo para HPG Aligner ARN BWT+M mejorado con la estructura metaexón

A continuación, se describe el funcionamiento de las etapas que forman los diferentes flujos de trabajo, además de ver cómo se actualiza la información de la estructura metaexón [39].

Flujo de trabajo 1. El flujo de trabajo 1 se divide en un total de cinco etapas, etiquetadas de la A a la F en la Figura 3.11a. Las operaciones realizadas en cada una de estas etapas son similares a las del método anterior. La Figura 3.9 muestra el diagrama de decisiones para el procesamiento de una lectura en el primer flujo de trabajo.

Etapa A (lectura de datos). Esta etapa lee los datos de entrada formando paquetes de «n» lecturas y los almacena en la cola «Read queue» de la Figura 3.11a.

Etapa B (BWT). Busca cada una de las lecturas en el genoma de referencia mediante nuestra propia implementación de BWT. Si la lectura mapea correctamente, el alineamiento se almacena y se actualiza la estructura metaexón con esta información. Si por el contrario, no se consigue mapear la lectura, BWT devuelve las posiciones hasta donde ha conseguido alinearla. En este caso, si los dos extremos son alineados una cantidad de nucleótidos suficientemente grande, se considera una lectura con dobles anclajes. Por el contrario, si solo es un extremo, se considera una lectura con un simple anclaje. No obstante, ambos tipos de lecturas, junto con las no mapeadas, pasarán a la siguiente etapa de procesamiento.

Etapa C (regiones y buscador de CAL). La etapa C de esta implementación corresponde a las etapas C y D del método anterior. Esta etapa fragmenta las lecturas formando semillas. Además, mapea las semillas para obtener las regiones candidatas de mapeo (CALs). Esta modificación reduce considerablemente el movimiento de datos entre las dos etapas, favoreciendo la localización de la información.

Etapa D (SWA + búsqueda de uniones). Esta etapa procesa tres tipos diferentes de entradas: las lecturas con dobles/simples anclajes y las lecturas con CALs. El procesamiento es distinto según el tipo de lectura.

Si la entrada es una lectura con dobles anclajes, en primer lugar se comprueba la distancia entre estos. Si esta distancia es inferior al tamaño mínimo de intrón, un algoritmo heurístico alinea el hueco comprendido entre ambos anclajes. Si se completa con éxito, la lectura se almacena como mapeada y la estructura metaexón se actualiza. En caso contrario, la lectura se almacena como no mapeada. En cambio, si la distancia está comprendida entre el tamaño mínimo y máximo de intrón, se intenta buscar un punto de unión entre exones. En primer lugar, se intenta encontrar este punto de unión mediante la estructura metaexón. En caso de no conseguirlo, se realizan las siguientes acciones:

(i) Si el hueco entre ambos anclajes es mayor que el tamaño de la semilla, se procede a aplicar el mismo procedimiento que en la etapa C: realizar semillas del hueco y obtener las CALs correspondientes.

(ii) Usando la información de los anclajes y las CALs obtenidas, en el caso de existir, se realiza una simple búsqueda en el genoma de referencia buscando las marcas de intrón («GT-AG» o «CT-AC»).

(iii) Si esta búsqueda falla, a continuación se aplica una búsqueda mediante el algoritmo SWA para alinear el hueco frente al genoma de referencia e identificar el punto de unión.

(iv) Si la lectura se mapea correctamente, las estructuras metaexón y AVL se actualizan. En caso contrario, la lectura se almacena como no mapeada.

Cuando la entrada de la etapa es un simple anclaje, se consulta la estructura metaexón para intentar mapear la lectura. Si se encuentra un mapeo, la estructura metaexón no se actualiza, ya que el alineamiento se considera poco fiable. Si por el contrario, la lectura no se puede mapear, se añade al fichero «SINGLE ANCHOR» de la Figura 3.11a.

Finalmente, cuando la entrada de la etapa es un conjunto de CALs, primero se comprueba la estructura metaexón para cada CAL o grupo de CALs para completar los alineamientos. Si no se consigue ninguno, se llevan a cabo las acciones descritas para esta etapa según la implementación anterior del algoritmo.

Al finalizar esta etapa, se almacenan los alineamientos de las lecturas mapeadas completamente, finalizando su procesamiento. Por el contrario, si en alguno de los dos últimos

casos alguno de sus extremos no se ha conseguido alinear correctamente, la lecturas se añaden al fichero «HARD CLIPPING» (Figura 3.11a). Estas lecturas se procesarán en el último flujo de trabajo.

Etapa F (escritura de resultados). Esta etapa escribe los alineamientos obtenidos en el fichero de salida en formato BAM/SAM.

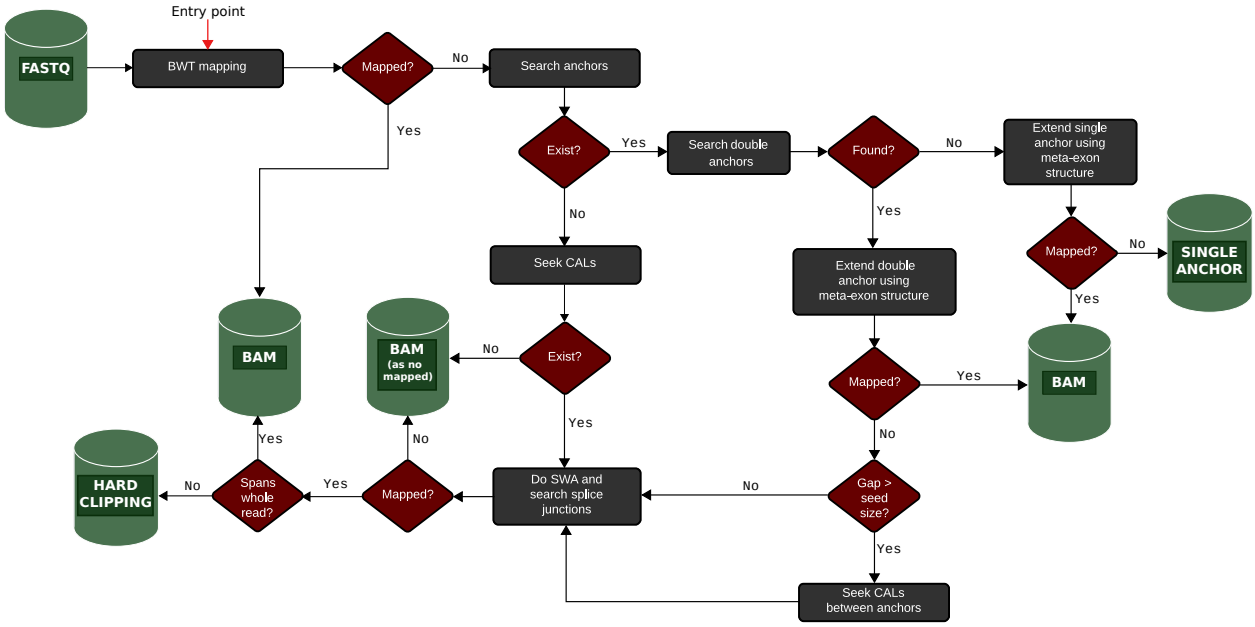


Figura 3.9: Diagrama de decisiones del procesamiento de una lectura para HPG Aligner ARN BWT+M en el flujo de trabajo 1

Flujo de trabajo 2. El segundo flujo de trabajo (Figura 3.11b), procesa las lecturas almacenadas en el fichero etiquetado como «SINGLE ANCHOR» en la Figura 3.11b. Este flujo de trabajo está formado por tres etapas de procesamiento, y también se ayuda de la estructura metaexón. En la Figura 3.10 se muestra el diagrama de decisiones por el que transcurre una lectura durante su procesamiento.

Etapa A (lectura de datos). En esta etapa se crean los paquetes de lecturas con los simples anclajes a partir del fichero «SINGLE ANCHOR».

Etapa B (procesamiento de las lecturas). Inicialmente se intenta extender la parte mapeada de la lectura con ayuda de la estructura metaexón. Si esto no es posible, se realizan semillas de la lectura para formar CALs y aplicar el algoritmo SWA para intentar encontrar algún alineamiento. Si esto ocurre, se actualiza la estructura metaexón, se almacenan los alineamientos y los puntos de unión (en caso de existir) en la estructura AVL. Si se encuentran CALs y no se consigue alinear la lectura esta se almacena en el fichero «HARD CLIPPING» de la Figura 3.11b. Si por el contrario, no se encuentran CALs, la lectura se almacena el fichero de salida como no mapeada.

Etapa C (escritura de resultados). Finalmente, se almacenan los resultados obtenidos por el procesamiento en el fichero de salida.

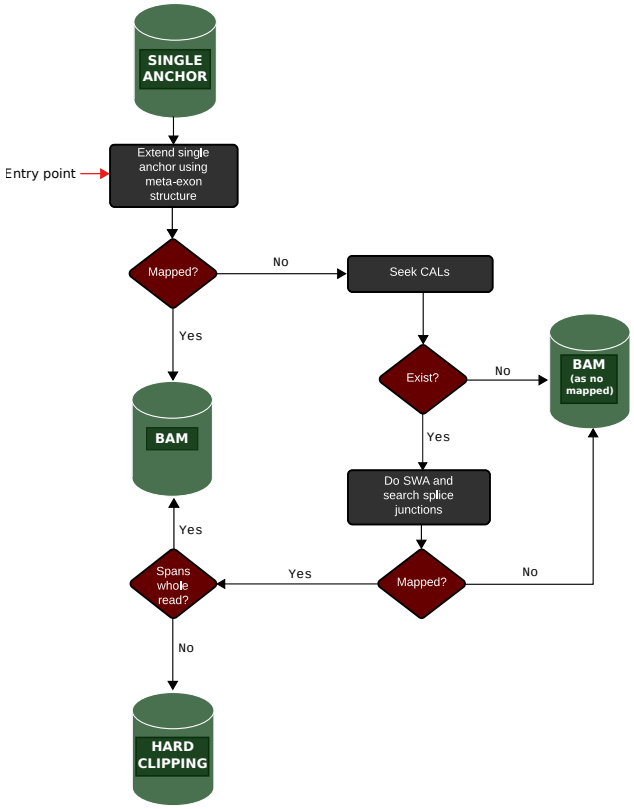


Figura 3.10: Diagrama de decisiones del procesamiento de una lectura para HPG Aligner ARN BWT+M en el flujo de trabajo 2

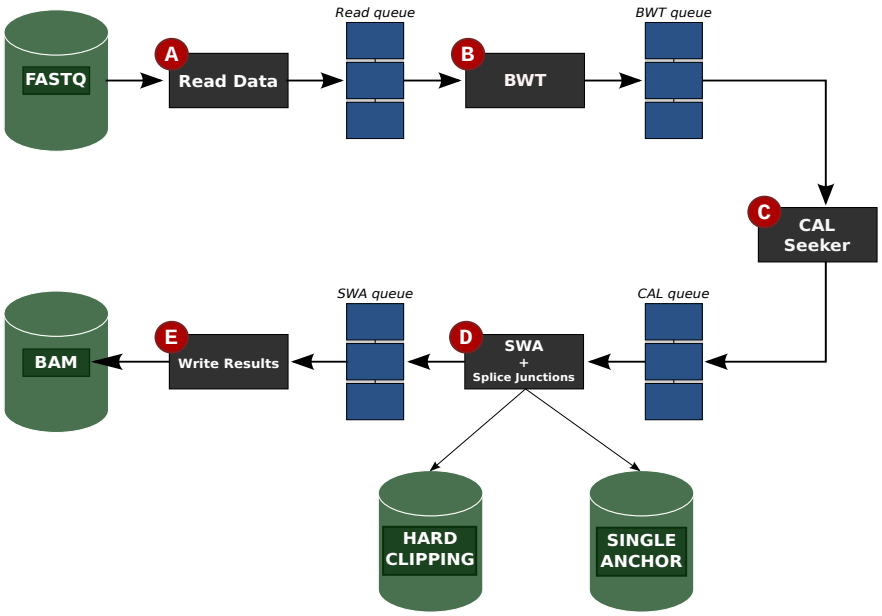
Flujo de trabajo 3. El tercero y último flujo de trabajo (ver Figura3.11c) procesa las lecturas del fichero «HARD CLIPPING» de la Figura 3.11c. Estas lecturas son las que no se han podido mapear por completo en los flujos de trabajo previos. La Figura 3.12, muestra el diagrama de decisiones para una lectura durante la ejecución del tercer flujo de trabajo.

Etapa A (lectura de datos). Esta etapa genera los paquetes de lecturas con la parte del alineamiento encontrado hasta el momento.

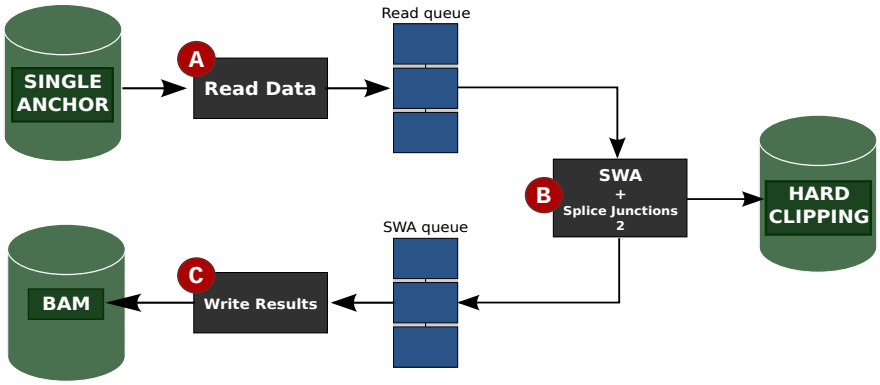
Etapa B (procesamiento de las lecturas). En esta etapa se procesan las lecturas del fichero de entrada. El procesamiento consiste en comprobar para cada lectura si se ha conseguido alinear parte de esta en alguna de las etapas anteriores. En caso afirmativo, se utiliza la estructura metaexón para completar el alineamiento. En caso contrario, o si no se ha podido completar el mapeo en el paso anterior, se aplica el algoritmo SWA ayudándose de la estructura metaexón.

Etapa C (escritura de resultados). Finalmente, se almacenan en el fichero de salida los alineamientos obtenidos durante la etapa B.

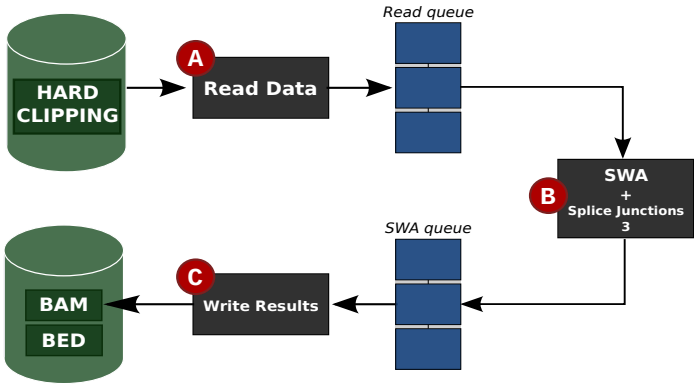
Por último, una vez todos los paquetes de lecturas han sido procesados y los resultados almacenados en el fichero de salida, los puntos de unión encontrados se almacenan en el fichero BED.



(a) Flujo de trabajo 1



(b) Flujo de trabajo 2



(c) Flujo de trabajo 3

Figura 3.11: Esquema de los tres flujos de trabajo que forman HPG Aligner ARN BWT+M

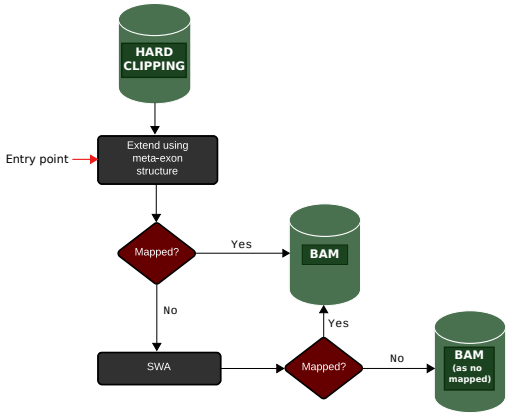


Figura 3.12: Diagrama de decisiones del procesamiento de una lectura para HPG Aligner ARN BWT+M en el flujo de trabajo 3

3.3.3 Implementación distribuida basada en la estructura metaexón

En la actualidad, gran parte de los centros de investigación disponen de clústeres de computadores. El método HPG Aligner ARN BWT+M+MPI [34] es una versión distribuida del método anterior que es capaz de explotar los recursos de este tipo de sistemas para acelerar al máximo el procesamiento. Este nuevo método, además de permitir que los nodos del clúster puedan trabajar de forma concurrente, también ha de asegurar que el reparto de la carga de trabajo entre los distintos nodos sea lo más equilibrada posible. Como el nombre de este método indica, la comunicación entre los nodos se ha implementado utilizando MPI, que es una interfaz de comunicación entre nodos basada en el paso de mensajes.

En esta nueva implementación se definen tantos procesos MPI trabajadores como nodos tiene el sistema. De esta manera, cada uno de estos procesos ejecuta el algoritmo descrito anteriormente sobre una parte diferente del fichero de entrada. Además, se crea un proceso adicional encargado de escribir en el fichero de salida los alineamientos encontrados durante el procesamiento.

El reparto de la carga entre los procesos se hace de forma estática. Se ha utilizado esta opción ya que en un estudio experimental previo se comprobó que el tiempo de procesamiento de todos los paquetes era muy similar. Así pues, antes de empezar con el procesamiento, el proceso 0 divide y reparte el trabajo entre el resto de trabajadores. Para esto, calcula el tamaño total del fichero de entrada, y a continuación, lo divide en partes iguales calculando las posiciones de inicio y fin de cada fragmento. Finalmente, este proceso envía esta información al resto de procesos para que cada uno de ellos lea su parte y la procese. Esta implementación multilector resulta más óptima que la versión de un único lector. En el caso de un único lector, este tendría que enviar las lecturas del fichero de entrada al resto de procesos mediante paso de mensajes. Esto conllevaría un consumo de tiempo más elevado en comparación con la versión propuesta, en la que varios procesos leen del mismo fichero al mismo tiempo.

Cuando un proceso trabajador recibe las posiciones del fragmento del fichero de entrada que tiene que procesar, lanza el alineador HPG Aligner ARN BWT+M. Tal y como se describió, este alineador se compone de tres flujos de trabajo secuenciales. Además, actualiza durante todo el procesamiento la estructura metaexón y el árbol AVL para incrementar la sensibilidad del algoritmo.

Por lo tanto, en esta nueva implementación, los nodos procesan en paralelo y de forma independiente los tres flujos de trabajo. Durante su procesamiento, van actualizando la estructura

metaexón y el árbol AVL. Esto implica que cada vez que finaliza un flujo de trabajo, todos los procesos se sincronizan y comparten la información recogida. De esta forma, todos trabajan con la misma información y se evita la pérdida de sensibilidad. Para llevar a cabo la unión de resultados, cada uno de los procesos envía sus estructuras de datos al nodo adyacente. Este nodo recogerá los datos y los unirá para posteriormente, volverlos a enviar. Este proceso se repite hasta que el nodo 0 tenga la unión de todas las estructuras de datos. La Figura 3.13 muestra un ejemplo del proceso de unión para un caso de 12 nodos. Finalmente, el nodo 0 envía las estructuras de datos actualizadas al resto de procesos para compartir la información recogida.

En esta implementación del algoritmo, a diferencia de las implementaciones anteriores, la etapa de escritura de cada flujo de trabajo guarda los resultados en un almacenamiento temporal. Cuando este se llena, envía los resultados al proceso escritor localizado en el nodo 0. Este proceso será el encargado de escribir los resultados obtenidos por el resto de procesos en el fichero de salida.

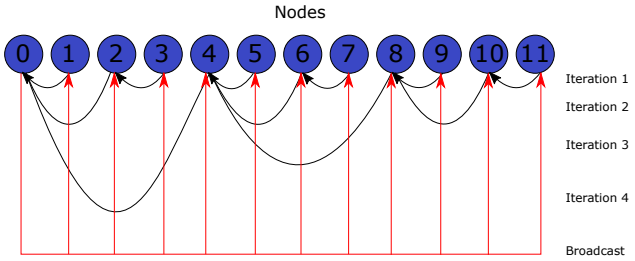


Figura 3.13: Diagrama de unión de las estructuras de datos

3.4 Método basado en la matriz de sufijos

En este apartado se describe un nuevo método para el alineamiento de secuencias de ADN y ARN. Este método aplica la técnica de búsqueda basada en una matriz de sufijos (SA) para realizar la búsqueda de las secuencias sobre el genoma de referencia. Este nuevo método combina la velocidad de utilizar las estructuras descomprimidas de la técnica SA, junto con la sensibilidad del algoritmo de Smith-Waterman (SWA). Esta nueva implementación está enfocada a sistemas multiprocesador siguiendo la estrategia de la asignación dinámica de tareas a hilos, tal y como se ha descrito en los apartados anteriores.

La técnica de búsqueda basada en una matriz de sufijos (SA) consiste en crear una matriz con todos los sufijos del texto en el que se quiere realizar la búsqueda (en nuestro caso el genoma de referencia), para después poder localizar rápidamente todas las apariciones de la cadena buscada. Para construir la matriz de sufijos se procede de la siguiente forma. En primer lugar, se añade el texto completo a la matriz; a continuación, se añade el texto completo salvo el primer carácter; después, el texto completo menos los dos primeros caracteres... Mientras se construye la matriz, cada sufijo se etiqueta con su posición de inicio en el texto completo. Una vez completado la matriz de sufijos, este se ordena alfabéticamente. Una vez ordenado, es posible buscar rápidamente todas las apariciones de una determinada cadena sin más que localizar todos los sufijos que comienzan por esa cadena. Puesto que la matriz está ordenada, estos sufijos estarán contiguos dentro de un intervalo de la matriz y pueden encontrarse eficientemente por medio de dos búsquedas binarias. La primera para localizar la posición inicial del intervalo y la segunda para determinar la posición final.

Utilizando esta matriz de sufijos es posible buscar en qué posiciones del genoma de referencia se encuentra una determinada secuencia. En el caso de HPG Aligner ADN/ARN SA, se utilizará está técnica para encontrar las semillas que se han obtenido a partir de las lecturas de ADN/ARN en el genoma de referencia. Hay que tener en cuenta que puesto que para cada lectura se generan varias semillas, se acabaran realizando millones de búsquedas. Por lo tanto, será necesario optimizar el método de búsqueda en la matriz de sufijos para reducir su coste computacional.

3.4.1 Optimización de la búsqueda en la matriz de sufijos mediante LCP y CRS

En HPG Aligner ADN/ARN SA se ha optimizado la búsqueda en la matriz de sufijos usando la técnica LCP (*Longest Common Prefix*). Esta técnica consiste en la implementación de una tabla *hash* que contiene los prefijos comunes por los que empiezan los sufijos. En concreto, en esta implementación, los prefijos consisten en todas las posibles combinaciones de 16 nucleótidos que pueden formarse. Por lo tanto, las entradas de la tabla *hash* contienen todas las posibles secuencias de nucleótidos. Por otro lado, cada una de estas secuencias tiene asociado un valor. Este valor indica la posición del primer sufijo que empieza por esta secuencia en la matriz de sufijos; o si el valor es nulo, que ningún sufijo empieza por esta secuencia. Por otra parte, para reducir el espacio en memoria, en lugar de almacenar los prefijos en forma de cadenas, estos se codifican de la siguiente forma. Como se ha visto anteriormente, existen cuatro tipos diferentes de caracteres, «A», «C», «G» y «T». Esto implica que cada carácter puede codificarse con solo 2 bits. Por lo tanto, como las cadenas están formadas por 16 nts, se necesitan 32 bits para codificarlas. Esta codificación permite obtener un valor numérico para cada cadena contenida en la tabla *hash* y, de esta forma, se almacena este valor en vez de la cadena completa.

La búsqueda de una secuencia se realiza inicialmente sobre la tabla *hash*. Para esto, en primer lugar, se codifica la cadena de 16 nts y se accede directamente al elemento correspondiente de la tabla *hash*. El contenido de esta entrada mostrará la posición de la matriz de sufijos a partir de la que se debe empezar a buscar esta secuencia de forma secuencial. De esta forma, se reduce el tiempo asociado a la búsqueda dicotómica.

El principal inconveniente del método propuesto son los requisitos de memoria necesarios para almacenar la tabla *hash*. En el caso descrito anteriormente, con semillas de 16 nts, se necesitarán 16 Gibibytes. Este espacio aumenta considerablemente cuando el tamaño de la semilla también lo hace. En concreto, si las semillas fueran de 18 nts, esta tabla ocuparía 512 Gibibytes. Utilizar semillas de mayor longitud proporcionaría una mayor velocidad de procesamiento, pero tal y como se ha visto, requiere de necesidades de memoria muy elevadas.

En HPG Aligner ADN/ARN SA, cuando el tamaño de semilla es de 18 nts se aplica la técnica de compresión de matrices CSR para reducir el espacio ocupado en memoria y hacer viable este método de búsqueda. Esta técnica se utiliza sobre matrices que tienen un gran número de elementos nulos. Como es fácil intuir, al aumentar el tamaño de semilla, el número de combinaciones que se obtienen crece considerablemente, mientras que el número de ocurrencias de cada semilla disminuye, y de hecho, muchas de las posibles secuencias de nucleótidos no estarán presentes en el genoma. Esto implica que muchas de las entradas de la tabla *hash* estarán vacías. Por lo tanto, si se convirtiera esta tabla en una matriz, se obtendría una matriz dispersa y se podría aplicar la técnica CSR para almacenarla de una forma más compacta. Para convertir la tabla *hash* en una matriz, se dividen los 64 bits que codifican las semillas en dos bloques, donde los primeros 56 bits pasan a indicar la fila de la matriz y los 8 bits restantes, la columna. Naturalmente, el contenido de la matriz es el mismo que el de la tabla *hash*: la posición en la matriz de sufijos del primer sufijo que comienza por esa secuencia. A continuación, a esta matriz se le aplica la técnica CSR generando de esta forma 3 vectores diferentes:

Vector A. Almacena los valores no nulos que se encuentran en la matriz. Es decir, las posiciones en el vector de sufijos de cada uno de los prefijos.

Vector IA. Tiene tantas entradas como número de filas de la matriz más una. Todas las entradas, salvo la última, se corresponden con las filas de la matriz original y contienen la posición en el vector «A» del primer elemento no nulo de esa fila. La última entrada contiene el total de elementos no nulos.

Vector JA. Es del mismo tamaño que el vector «A». Contiene el índice de columna de la matriz en la que se encuentra cada uno de los valores del vector «A».

De esta forma, solo se almacenan los valores no nulos de la matriz, reduciendo el tamaño que esta ocupa en memoria.

Mediante los tres vectores anteriores, es posible acceder a los valores de la matriz sin problemas. El proceso de búsqueda de una semilla se realiza en varios pasos. En primer lugar, se codifica la semilla según se ha descrito anteriormente. A continuación, los primeros 56 bits del valor indican la fila de la matriz, y los 8 restantes, la columna. Por lo tanto, primero se accede a la posición del vector «IA» fijada por los primero 56 bits de la semilla codificada. A continuación, se accede a la posición del vector «JA» indicada por el vector «IA». Finalmente, se recorre este vector a partir de esta posición hasta encontrar o superar el valor determinado por los últimos 8 bits de la semilla codificada. La posición de este valor dentro del vector señala a qué posición del vector «A» hay que acceder para encontrar el valor correspondiente de la matriz. Si durante la búsqueda en el vector «JA» el valor de alguna de las posiciones supera el valor fijado por los 8 bits de la semilla, se aborta la búsqueda, ya que esto significa que se trata de un valor nulo.

3.4.2 Algoritmo para el alineamiento de lecturas de ADN basado en SA

A continuación se describe el algoritmo para el alineamiento de secuencias de ADN denominado HPG Aligner ADN SA [48]. Al igual que el algoritmo para el alineamiento de lecturas de ARN, el procesamiento se ha dividido en una serie de etapas secuenciales. Además, sigue el mismo modelo de productor-consumidor, donde el conjunto de lecturas a mapear se divide en una serie de paquetes de tamaño fijo, que se almacenan en la cola de lectura. A diferencia del alineador de lecturas de ARN, cuando un hilo coge un paquete, no lo deja en la cola de escritura hasta que no finaliza por completo su procesamiento. Esto significa que todos los hilos aplican todas las etapas del procesamiento a un paquete. Esta modificación se ha realizado para mejorar la localidad de los datos y eliminar gran parte de los movimientos de información entre la cola y los hilos. Una vez el procesamiento del paquete finaliza, el hilo deja el paquete con la información procesada en la cola de escritura. Finalmente, el hilo escritor coge el paquete de esta cola y almacena los resultados en el fichero de salida.

Etapa A (lectura de datos). Esta etapa, al igual que las implementaciones descritas anteriormente, forma paquetes de lecturas a partir del fichero de entrada. Estos paquetes se almacenan en la cola de lectura para ser procesados a continuación. Además, esta etapa también permite leer ficheros de entrada comprimidos en formato gzip, sin tener la necesidad de descomprimirlos previamente.

Etapa B (semillas de las lecturas). Inicialmente, esta etapa coge los paquetes de la cola de trabajo pendientes. A continuación, forma semillas de 18 nucleótidos de longitud a partir de las lecturas de estos paquetes. Estas semillas cubren de forma uniforme toda la lectura.

Cada una de estas semillas se busca en la matriz de sufijos siguiendo los pasos explicados anteriormente. Al final del procesamiento, se habrá obtenido para cada semilla todas sus localizaciones genómicas.

Etapa C (extensión de semillas y obtención de CALs). Cada semilla mapeada se extiende, tanto por la izquierda como por la derecha, hasta alcanzar el número máximo de errores permitidos. A continuación, para cada lectura, se forman las CALs agrupando las semillas que han mapeado en regiones próximas en el genoma. Finalmente, se seleccionan las CALs que mejor cubren la lectura.

Etapa D (SWA). El último paso del procesamiento aplica nuestra implementación del algoritmo SWA a los huecos que no se han conseguido cubrir de la lectura. Además, se calcula la puntuación para cada uno de los alineamientos siguiendo los parámetros fijados por el usuario. Cuando ha finalizado por completo todo el procesamiento, esta etapa almacena el paquete en la cola de escritura.

Etapa E (escritura de resultados). Esta etapa coge los paquetes almacenados en la cola de escritura y almacena los alineamientos encontrados en el fichero de salida. Junto al alineamiento de estas lecturas se almacenan una serie de parámetros opcionales de ayuda para los usuarios, como es la puntuación del alineamiento, el número de errores de la lectura, etc. Los resultados pueden escribirse en formato BAM/SAM, al igual que en el procesamiento de ARN. En la Figura 3.14, puede observarse un ejemplo de alineamientos encontrados durante el procesamiento.

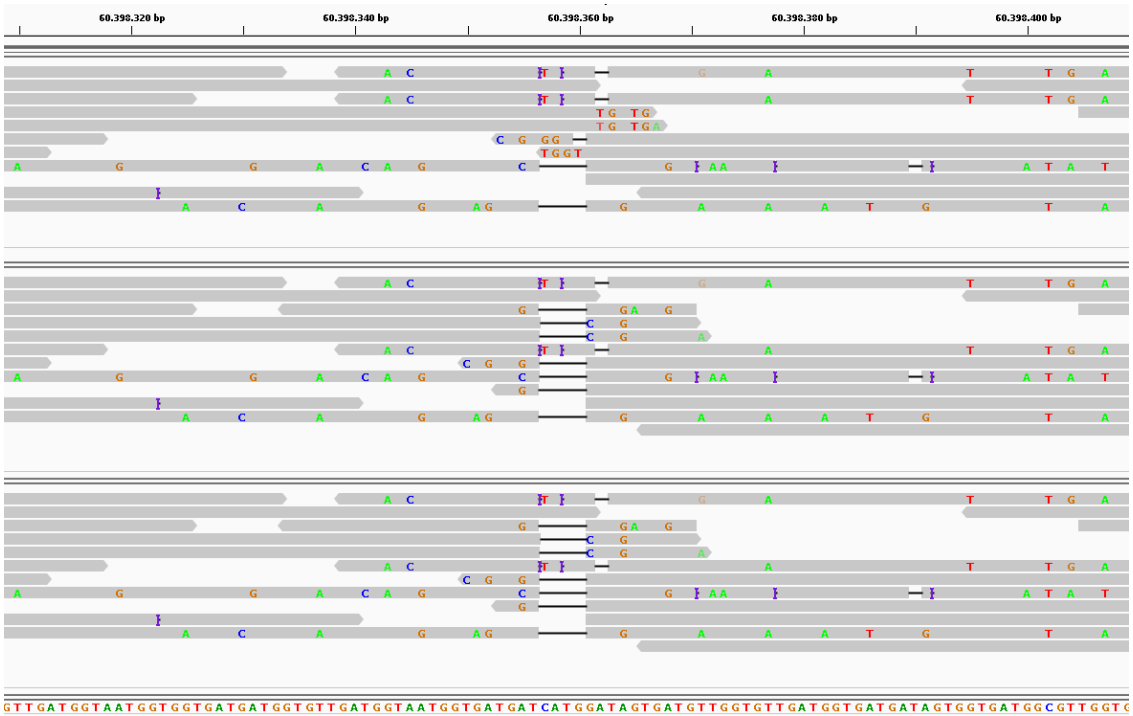


Figura 3.14: Ejemplo de alineamientos de lecturas encontrados por HPG Aligner ADN SA

3.4.3 Algoritmo para el alineamiento de lecturas de ARN basado en SA

En este apartado se describe la nueva implementación del algoritmo HPG Aligner ARN SA+M [37] basado en la matriz de sufijos [31] y la estructura metaexón. La diferencia fundamental respecto a la implementación de HPG Aligner ARN BWT+M, es la técnica para la búsqueda de secuencias en el genoma de referencia. Esta nueva técnica basada en SA, trabaja con el genoma de referencia sin comprimir. Esto implica, que el espacio que ocupa en memoria principal es mucho más elevado, pero a cambio, las búsquedas de secuencias en el genoma de referencia son más rápidas. Además de esto, se han modificado las operaciones realizadas por varias etapas y se han reorganizado los flujos de trabajo, pasando de tres flujos a dos. Todas estas modificaciones han generado un incremento de la velocidad de mapeo y de la sensibilidad respecto a las versiones anteriores del alineador.

Tal y como se ha descrito, HPG Aligner ARN SA+M consta únicamente de dos flujos de trabajo, tal y como muestra la Figura 3.18. El primero de ellos mapea todas las lecturas utilizando la técnica de SA y SWA, y actualiza la estructura metaexón con la información de los alineamientos. Las lecturas que al final de este procesamiento les falta por mapear alguna pequeña parte de sus dos extremos se almacenan en el fichero «HARD CLIPPING» de la Figura 3.18 para su procesamiento en el segundo flujo de trabajo. Finalmente, este segundo flujo de trabajo, con ayuda de la estructura metaexón y del algoritmo SWA, intenta completar los alineamientos de estas lecturas.

El motivo por el que ha desaparecido el fichero «SINGLE ANCHOR» y el flujo de trabajo correspondiente, es debido a que este tipo de lecturas son exclusivas de BWT.

Como se muestra en la Figura 3.17a, al igual que en la implementación para ADN, cuando un hilo coge un paquete de la cola de trabajos pendientes (etiquetada como «Read queue» en la Figura 3.17a) lo procesa por completo. Cuando el hilo finaliza el procesamiento del paquete, lo almacena en la cola de escritura (etiquetada como «Write queue» en la Figura 3.17a). Esto significa, que las colas intermedias entre las etapas B, C y D han desaparecido. Estas modificaciones mejoran la localidad de datos y reducen los tiempos de sincronización entre hilos. En la Figura 3.15 se observa una traza del primer flujo de trabajo de este algoritmo. En esta traza se muestra que todos los hilos están trabajando durante todo el transcurso de la ejecución. Por otro lado, en la traza puede comprobarse como existe únicamente un lector y un escritor al mismo tiempo, evitando problemas de lectura y de escritura. A continuación se describe con detalle el procesamiento que se lleva a cabo en cada flujo de trabajo.

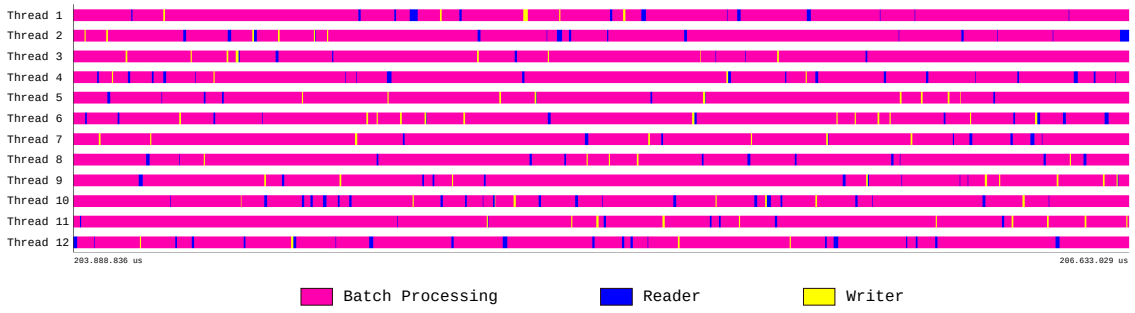


Figura 3.15: Traza de la ejecución del flujo de trabajo 1 con 12 hilos/núcleos. Cada barra horizontal muestra en cada momento el tipo de acción que realiza el hilo (lectura/escritura/procesamiento)

Flujo de trabajo 1. Este flujo se divide en cinco etapas de procesamiento como se puede ver en la Figura 3.17a. En la Figura 3.19 puede observarse el diagrama de decisiones por el que transcurre una lectura a través de este flujo de trabajo.

Etapa A (lectura de datos). Esta etapa forma paquetes de lecturas a partir del fichero de entrada. Esto paquetes se almacenan en la cola «Read queue» de la Figura 3.17a. Al igual que el resto de implementaciones, esta etapa es capaz de leer los datos de entrada comprimidos en formato gzip.

Etapa B (matriz de sufijos). Inicialmente, esta etapa coge los paquetes de lecturas de la cola «Read queue» de la Figura 3.17a. A continuación, mapea las lecturas del paquete aplicando la técnica SA. Para esto, se toman los primeros 18 nucleótidos de la lectura y se busca este fragmento en la matriz de sufijos. Todos los mapeos encontrados se intentan extender hasta cubrir por completo la lectura. Si esto ocurre, la lectura se considera mapeada, se actualiza la estructura metaexón y los alineamientos se almacenan en el paquete. Si no se consigue extender ningún mapeo hasta cubrir toda la lectura, esta se considera no mapeada y se procesará en la siguiente etapa.

Etapa C (semillas dinámicas + búsqueda de CAL). Esta etapa, en primer lugar, forma semillas dinámicas a partir de las lecturas no mapeadas. Para esto, inicialmente forma fragmentos de 18 nts. Estos fragmentos se mapean con ayuda de la matriz de sufijos. Los mapeos encontrados se intentan extender incrementando el área alineada. Si una semilla no se consigue mapear, se realiza una nueva semilla que se solapa con esta en 8 nts, tal y como muestra la Figura 3.16. El siguiente paso consiste en buscar las CALs a partir de los mapeos obtenidos con estos fragmentos. Para esto, se aplica el mismo método que en las implementaciones anteriores.

Etapa D (SWA + búsqueda de uniones). Inicialmente, para cada lectura, se agrupan las CALs que se encuentran a una distancia inferior a la del tamaño máximo de intrón. A continuación, a cada CAL o grupo de CALs, se le asigna una puntuación en función del porcentaje de lectura cubierto. Este paso acelera el procesamiento y reduce los falsos positivos, ya que las CALs con una puntuación inferior a un umbral dado, se descartan.

En función del tamaño y localización de las zonas de la lectura no cubiertas por las CALs, se aplica una operación u otra. De esta forma, si los huecos de la lectura no están localizados en los extremos y su tamaño es similar al de los huecos de las CALs, se aplica el algoritmo SWA en estas zonas. Por otro lado, si el hueco entre las CALs es mucho mayor que la parte no mapeada de la lectura, se intenta buscar un punto de unión entre exones. Para esto, se extiende el hueco desde sus extremos hacia el centro buscando las marcas de intrón. Estas marcas son «GT-AG» y «CT-AC» para las uniones canónicas, y «AT-AC», «GT-AT», «GC-AG» y «CT-GC» para las semi-canónicas. Si no se consigue completar el hueco mediante esta extensión, se aplica el algoritmo SWA para buscar los puntos de unión. Para esto, se siguen los mismos pasos que en las implementaciones anteriores. Estos pasos consisten en extender las CALs 30 nucleótidos por ambos extremos y buscar los puntos de unión en los huecos grandes del alineamiento. Estos huecos, como ya se ha descrito anteriormente, se generan por la inclusión de los fragmentos de intrón cuando se extienden las CALs.

En cualquiera de los dos casos anteriores, si se consigue completar el alineamiento, la lectura se considera mapeada, y además se actualiza la estructura metaexón. Por el contrario, si no se consigue completar el alineamiento, la lectura se almacenará como no mapeada.

Finalmente, si los huecos se localizan en los extremos de las lecturas, se intentan completar con ayuda de la estructura metaexón. En caso de conseguirse, se actualiza esta estructura y la lectura se considera como mapeada. En caso contrario, la lectura se almacena en el fichero etiquetado como «HARD CLIPPING» en la Figura 3.17a.

Todos los puntos de unión encontrados durante esta etapa, se almacenan en la estructura AVL.

Etapa E (escritura de resultados). Esta etapa coge los paquetes almacenados en la cola «Write queue» de la Figura 3.17a. A continuación, almacena los alineamientos encontrados en el fichero de salida. A diferencia de las anteriores versiones del algoritmo, junto a cada alineamiento se adjunta una nueva información de ayuda para el usuario. Entre esta información, se encuentra la puntuación generada para el alineamiento. Esta puntuación se calcula mediante la siguiente fórmula:

$$score = \frac{(w_m \cdot matches - w_i \cdot mismatches \ - w_o \cdot indels_{open} - w_c \cdot indels_{close}) \cdot 100}{(w_m \cdot read\_length)}$$

donde w_m, w_i, w_o y w_c son fijados con los valores predeterminados del algoritmo Smith-Waterman (5, 4, 10, y 0,5, respectivamente). En cualquier caso, estos valores pueden ser cambiados por el usuario.

Flujo de trabajo 2 Este flujo de trabajo se divide en tres etapas de procesamiento como muestra la Figura 3.17b. Además, se encarga de procesar las lecturas almacenadas en el fichero etiquetado como «HARD CLIPPING» en esta figura. Estas lecturas son las que les falta por alinear alguno de sus dos extremos. En la Figura 3.20 puede observarse el diagrama de decisiones por el cual transcurre una lectura a través de este flujo.

Etapa A (lectura de datos). Esta etapa lee las lecturas, junto con sus alineamientos parciales, desde el fichero «HARD CLIPPING» de la Figura 3.20. Además, forma paquetes, al igual que el flujo de trabajo anterior, y los almacena en la cola «Read queue» de la Figura 3.20.

Etapa B (SWA + búsqueda de uniones). Esta etapa intenta completar los alineamientos encontrados para cada una de las lecturas. Para conseguir esto, inicialmente, se ayuda de la estructura metaexón. Si se consigue completar, la lectura se almacena como mapeada. En caso contrario, se aplica el algoritmo SWA sobre los extremos de la lectura no mapeados y la zona genómica correspondiente. Si la puntuación obtenida por el alineamiento supera el umbral fijado, la lectura se almacena como mapeada. Si no se consigue mapear, se divide la lectura en semillas y se intenta encontrar algún alineamiento junto con la estructura metaexón. Si de esta forma tampoco se consigue encontrar ningún alineamiento, esta lectura se almacena finalmente como no mapeada.

Etapa C (escritura de resultados). Esta etapa coge los paquetes de la cola «Write queue» de la Figura 3.20 y almacena sus alineamientos en el fichero de salida.

Una vez ha finalizado el procesamiento del flujo de trabajo 2 se escriben todos los puntos de unión entre exones en el fichero de salida BED. Como ya se ha descrito, estos puntos de unión se almacenan en una estructura AVL.

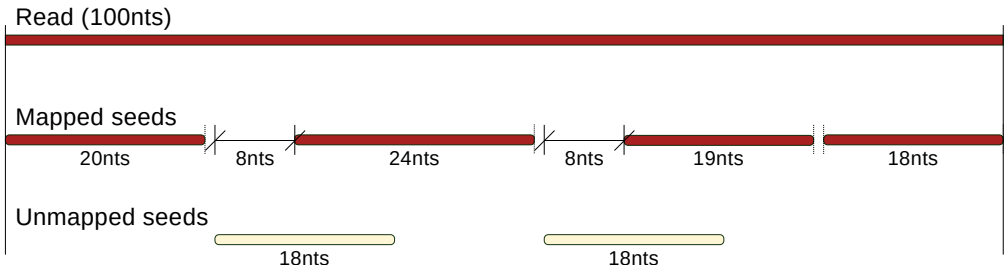
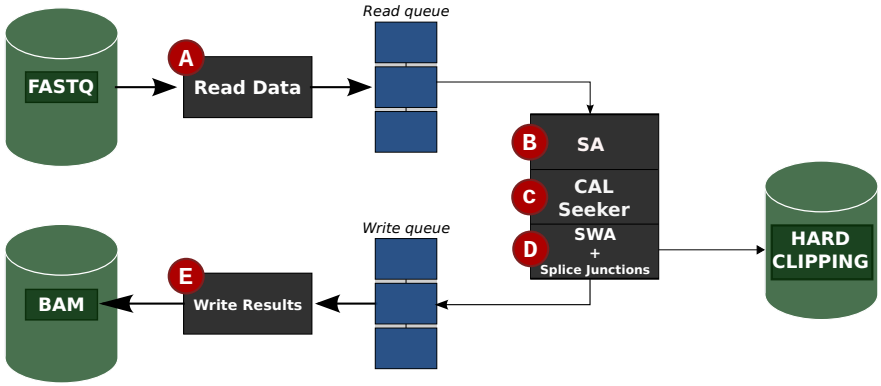
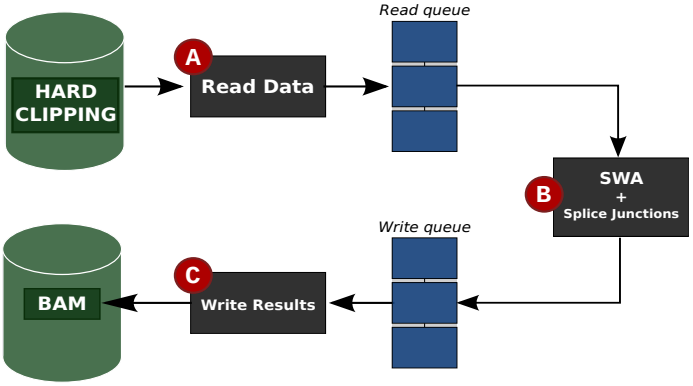


Figura 3.16: Ejemplo de fragmentación y extensión de las semillas de una lectura de 100 nt. Cuatro de las seis semillas generadas han sido mapeadas, y además, tres de ellas extendidas. Por el contrario, dos de las semillas no han sido mapeadas (parte inferior de la imagen)



(a) Flujo de trabajo 1



(b) Flujo de trabajo 2

Figura 3.17: Esquema de las etapas que forman parte de los dos flujos de trabajo de HPG Aligner ARN SA

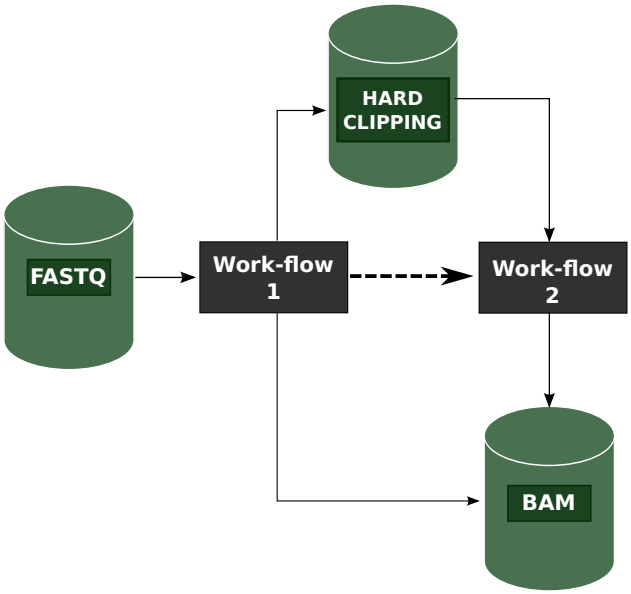


Figura 3.18: Diagrama de los flujos de trabajo de HPG Aligner ARN SA+M

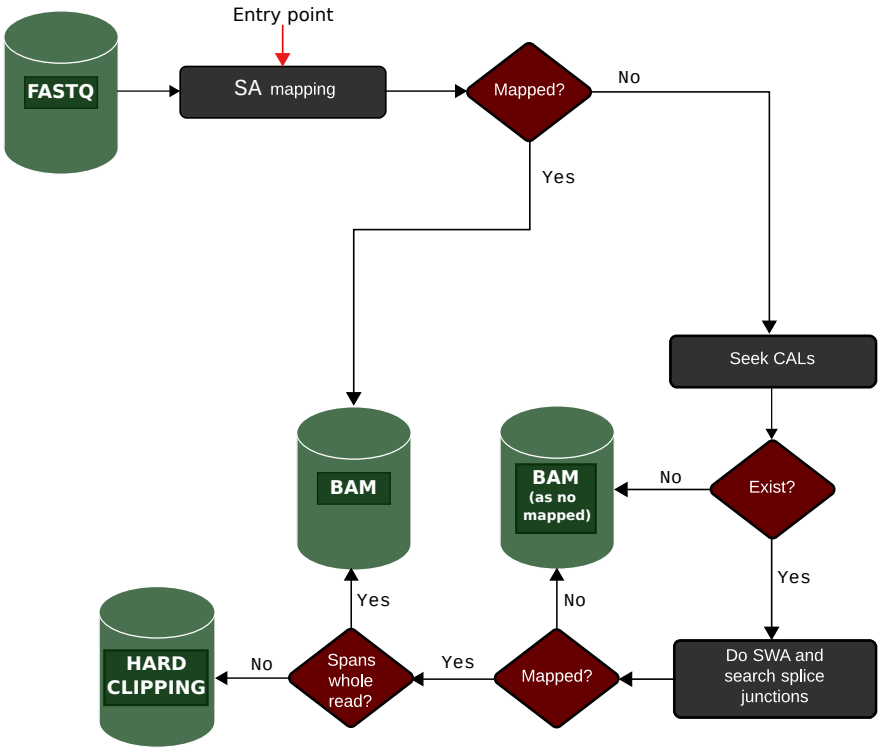


Figura 3.19: Diagrama de decisiones del procesamiento de una lectura para HPG Aligner ARN SA+M en el flujo de trabajo 1

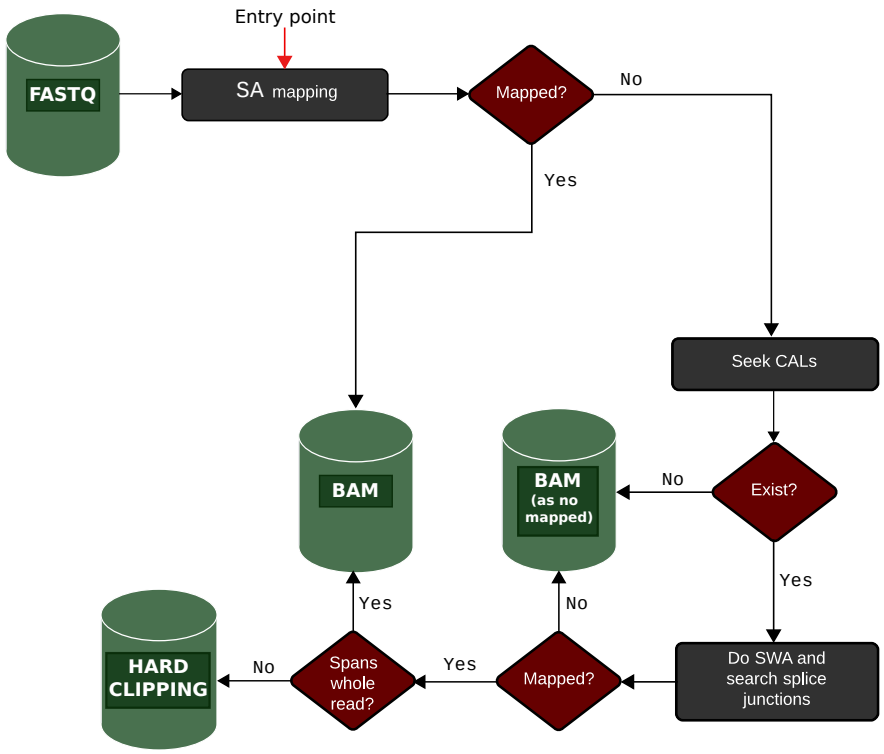


Figura 3.20: Diagrama de decisiones del procesamiento de una lectura para HPG Aligner ARN SA+M en el flujo de trabajo 2

3.5 *Framework*

En este apartado se presenta el alineador genérico *framework* [35] para secuencias de ARN y ADN. Este alineador permite ejecutar sobre un clúster de computadores cualquier alineador multihilo o secuencial sin la necesidad de modificar su código fuente. La clave de esta implementación es hacer transparente para el usuario el proceso de migración de un alineador cualquiera a un sistema clúster. Además, siguiendo el diseño del algoritmo HPG Aligner ARN BWT+M+MPI, el *framework* se ha organizado en dos etapas de procesamiento, permitiendo ejecutar diferentes alineadores en cada una de estas etapas. Esta característica permite combinar dos alineadores de tal forma que se obtenga un alineamiento más rápido que el que conseguiría uno de ellos y con una sensibilidad mejor que la que se conseguiría con el otro. Para ello, bastaría con utilizar en la primera etapa el alineador más rápido (con peor sensibilidad), con lo que se alinearían la mayor parte de las lecturas y, en la segunda etapa, un alineador con mejor sensibilidad (y seguramente más lento), que solo tendría que alinear las lecturas que no hubiera podido alinear el primero. De esta forma, se conseguiría un alineamiento con una sensibilidad mejor que si solo se hubiera utilizado el primer alineador y más rápido que si solo se hubiera utilizado el segundo.

En concreto, las principales características que presenta el *framework* son:

- Ofrece un esqueleto de ejecución para alineadores de ADN/ARN en sistemas multinodo. Entre otros aspectos, este se encarga de la distribución de la carga, el intercambio de información entre procesos y la unificación de resultados en un único fichero de salida.
- Gracias a las dos etapas que forman parte del procesamiento, ofrece la posibilidad de remapear las lecturas no mapeadas durante el procesamiento inicial. Estas etapas están conectadas a través de un almacenamiento temporal donde se almacenan las lecturas no mapeadas. La segunda etapa toma estas lecturas y realiza de nuevo el proceso de mapeo.
- El usuario es el encargado de seleccionar el/los alineadores que se ejecutarán en cada una de las etapas del *framework*. Al existir un gran número de alineadores y configuraciones donde elegir, el *framework* es una herramienta capaz de mejorar la sensibilidad y velocidad del procesamiento de lecturas de ADN/ARN.

Las principales funciones de este *framework* son:

- Crear los procesos MPI fijados por el usuario.
- Distribuir de forma equilibrada la carga de trabajo entre todos los procesos, es decir, dividir y enviar las lecturas a mapear a los distintos procesos.
- Lanzar la ejecución del alineador seleccionado por el usuario en los diferentes nodos del sistema. Este puede ser distinto en cada una de las etapas del *framework*.
- Si es necesario, crear los almacenamientos temporales de comunicación entre la primera y la segunda etapa.
- Implementar las comunicaciones entre los procesos MPI.
- Sincronizar la finalización de todo el procesamiento.

Aunque el punto de partida de la implementación del *framework* es HPG Aligner ARN BWT+M+MPI, ambos alineadores presentan importantes diferencias de diseño como:

- En cada proceso MPI del *framework* se ejecuta un alineador para lecturas de ADN/ARN como puede ser, Bowtie 2, STAR, MapSplice 2, etc.
- El *framework* tiene que dividir la carga de trabajo (las lecturas a procesar) en ficheros separados para cada uno de los procesos.
- Además, se encarga de recopilar y procesar los resultados generados por los alineadores en un único fichero de salida.
- Finalmente, tiene que ser capaz de procesar la salida de una gran variedad de alineadores utilizados en la primera etapa para producir información adecuada para pasarla a la segunda.

3.5.1 Interfaz y modo de operación del *Framework*

Para que el *framework* pueda ejecutarse y llevar a cabo sus funciones se tienen que especificar diferentes parámetros desde la línea de comandos:

- Número de procesos MPI que se crearán.
- Número de etapas que se ejecutarán (1 o 2).
- Tipo de secuencias del problema a procesar (ADN o ARN).
- Nombre de los alineadores que se usarán en cada etapa (junto con su configuración).
- Nombre de los ficheros de entrada que contienen el genoma de referencia y las lecturas para procesar.
- Nombre de los directorios para almacenar los resultados intermedios.
- Nombre del directorio y los ficheros de salida.

La Figura 3.21 muestra un ejemplo de lanzamiento donde se indica: el número de procesos MPI a generar, «n+1»; el nombre de los nodos del clúster donde se ejecutarán; el tipo de lecturas a procesar, «ARN»; el alineador utilizado en la primera etapa, «Tophat 2»; y su configuración, «-o %O... %I». La línea de comandos continúa mostrando: el nombre del fichero que contiene el genoma de referencia, «index_hpg»; el nombre del fichero que contiene las lecturas de entrada «freads.fq»; el nombre del fichero de salida y los ficheros temporales, «final-output/» y «/tmp/output». El último parámetro de esta línea indica si la segunda etapa del *framework* está activa o no, «–second-phase». En este caso concreto está activa, pero no se indica ningún mapeador, por tanto, se lanzará por defecto HPG Aligner ARN SA.

A partir de este lanzamiento se ejecuta el *framework*. En primer lugar, se crean un total de «n + 1» procesos sobre un total de «n» nodos del clúster. A continuación, el proceso trabajador 1 divide el fichero de entrada en «n» fragmentos iguales almacenándolos en ficheros diferentes. Después, cada proceso trabajador procesa uno de los nuevos ficheros generados mediante el alineador seleccionado por el usuario. Los resultados locales son almacenados en ficheros temporales en formato SAM. Este modo de funcionamiento se refleja en la Figura 3.22.

El procesamiento continúa con la ejecución de la segunda etapa, si esta se hubiera seleccionado. En el caso de que lo esté, en la segunda etapa se procesarán las lecturas no mapeadas y que están almacenadas en cada uno de los ficheros correspondientes a cada uno de los nodos, por tanto, no hará falta una nueva distribución de la carga. El procesamiento requerido en esta segunda etapa es distinto según el tipo de mapeo:

```
mpirun -np n ./hpg-multialigner rna -c "tophat2 -o \%O -p 16
  --no-convert-bam index\_bwt/homo\_sapiens \%I"
  -i index\_hpg/
  -f reads.fq
  -o final-output/
  --tmp-path /tmp/output/
  --second-phase
```

Figura 3.21: Ejemplo de línea de lanzamiento del *framework*

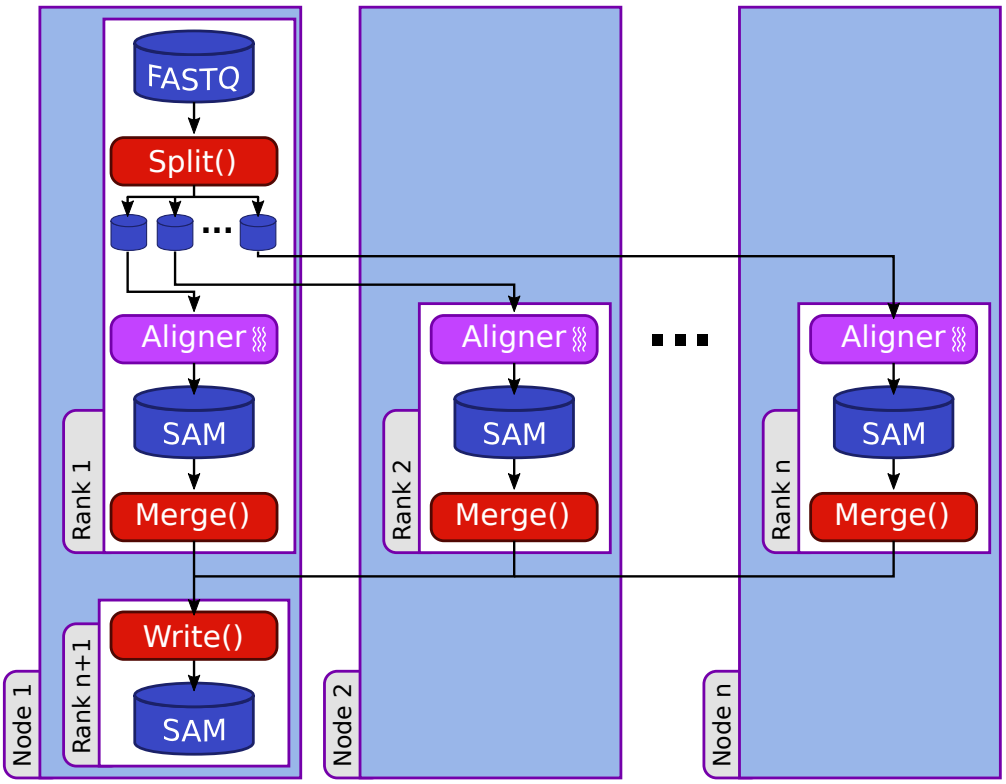


Figura 3.22: Diagrama de las etapas que forman el *framework*

ADN. En este caso, cada proceso trabajador analiza de forma local las lecturas no mapeadas durante la primera etapa de procesamiento. Esto significa, que no es necesario realizar ningún tipo de división de trabajo, ya que las lecturas se encuentran en los diferentes ficheros de salida generados.

ARN. Una vez finalizada la primera etapa de procesamiento, se obtienen todos los puntos de unión cubiertos por las lecturas mapeadas. Desafortunadamente, los alineadores almacenan esta información en ficheros de diferentes formatos. Por este motivo, el *framework* genera estos puntos de unión a partir de los alineamientos obtenidos y los almacena en un fichero con formato BED. En caso de que la segunda etapa haya sido activada, y sea HPG Aligner

ARN SA o HPG Aligner BWT+M el alineador seleccionado, se tiene que generar la estructura de metaexón para continuar con el proceso. Esta estructura se genera de forma local en cada uno de los nodos. A continuación, se juntan en una estructura global antes del procesamiento de la segunda fase. Finalmente, al igual que en ADN, se procesan las lecturas no mapeadas durante la primera etapa.

Una vez finalizada la primera y la segunda etapa, los resultados locales se enviarán al proceso escritor «n+1» que los unificará en un único fichero de salida en formato SAM.

3.5.2 Optimización del *framework*

La implementación original del *framework* presenta dos cuellos de botella: el primero se produce durante la distribución de la carga de trabajo y el segundo, en la mezcla de los resultados obtenidos tras finalizar el procesamiento. Como el tiempo requerido para llevar a cabo estas dos tareas es comparable al coste del procedimiento general del mapeo, la escalabilidad del *framework* se ve afectada.

Tal y como se ha descrito, la distribución de trabajo inicial es costosa porque se ha de leer por completo el fichero de entrada y reescribirlo en diferentes fragmentos de fichero. Este proceso es necesario ya que al alineador seleccionado por el usuario hay que indicarle cada uno de los nombres de los ficheros creados. Por otro lado, la unión de los resultados de cada etapa también supone un alto coste debido a que no se puede iniciar hasta que no haya finalizado el procesamiento por completo. Ambos procedimientos aseguran la ejecución correcta de una amplia variedad de alineadores a través del *framework* sin modificar el código original, pero provocan un incremento sustancial del tiempo de ejecución. Esta problemática no se produce en HPG Aligner BWT+M+MPI ya que por un lado, todos los trabajadores leen simultáneamente los datos del fichero de entrada y por otro, la combinación de los resultados se solapa con el procesamiento. Ambas mejoras pueden introducirse en el *framework*, aunque es considerablemente más difícil que en el caso del HPG Aligner ARN BWT+M+MPI, ya que no se tiene control sobre el código de los otros alineadores.

La optimización propuesta para el *framework* incluye ambas características de forma opcional. Desde el punto de vista de la interfaz, la opción de la línea de comando «–fifo-enable» le indica al alineador genérico que solape la división de trabajo con el procesamiento. Por otra parte, la opción de la línea de comando «–fast-merge» le indica que una los resultados solapándolos con el procesamiento. Cuando se seleccionan ambas opciones, el esquema de procesamiento se reorganiza como se muestra en la Figura 3.23.

En esta configuración, cada proceso MPI crea un hilo lector para realizar la distribución del fichero de entrada de forma paralela. Además, se crea un hilo de unión para paralelizar y solapar el proceso de unión de resultados. Para distribuir el fichero de entrada, uno de los procesos calcula las posiciones de inicio/fin que corresponden a cada trabajador y difunde esta información al resto. El hilo lector en cada trabajador crea un fichero FIFO (también conocido como tubería con nombre) en su RAM local y escribe su parte correspondiente del fichero original de entrada a la tubería. Tras la invocación, el nombre del fichero FIFO también se pasa al ejecutable del alineador (a través de su línea de comando), que simplemente recupera su entrada desde esa tubería. Procediendo de esta manera, el fichero de entrada se lee simultáneamente por los hilos lectores de cada trabajador, que distribuyen el conjunto de datos al mismo tiempo. Mediante este procedimiento no es necesario modificar el código fuente de los alineadores.

Como parte del proceso de combinación de resultados, el hilo de fusión en cada proceso espera hasta que se crea un fichero de salida local. Tan pronto como se detecta este evento, este

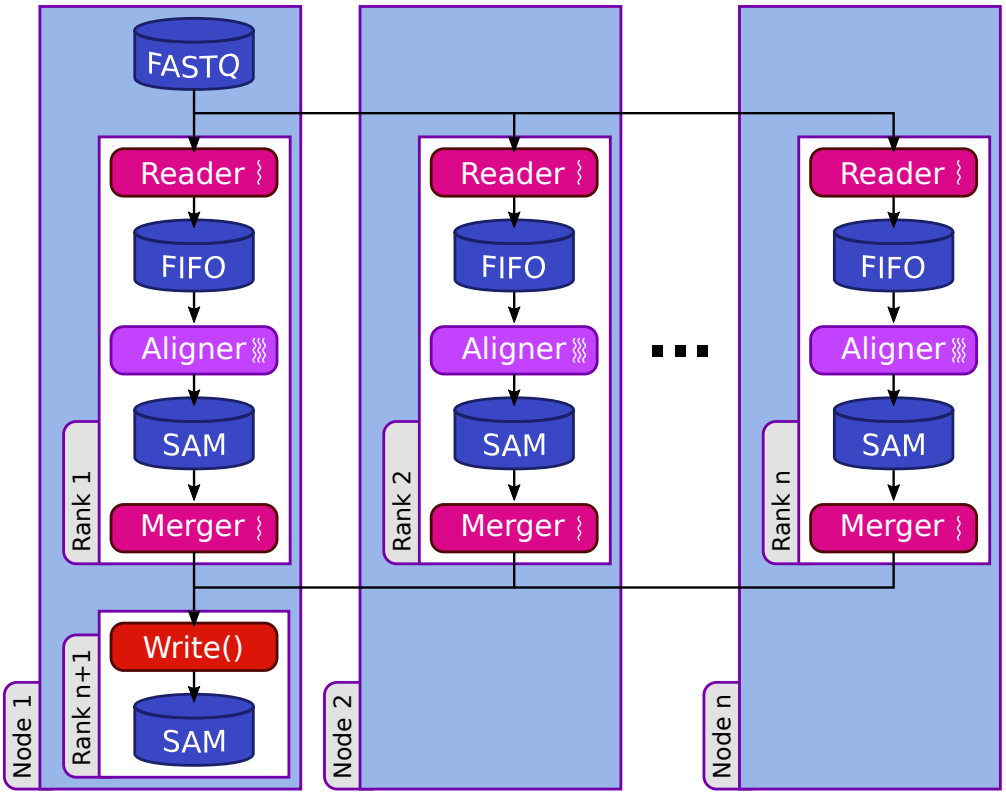


Figura 3.23: Diagrama de las etapas que forman el *framework* mejorado

hilo comienza a leer los resultados desde allí. Cuando tiene suficientes datos para llenar un búfer, envía esta información al proceso escritor global. La fusión continúa de esta manera hasta que el proceso de mapeo local haya finalizado, y la información contenida en los ficheros de salida locales se haya transmitido completamente al escritor global.

CAPÍTULO 4

Resultados experimentales

En este capítulo se se estudia el rendimiento de los alineadores implementados. El objetivo de este análisis es el de contrastar la sensibilidad, la especificidad y el rendimiento temporal de las aplicaciones más relevantes existentes actualmente para el alineamiento de secuencias de ADN/ARN frente a cada uno de los métodos desarrollados en este trabajo.

Estos estudios se han llevado a cabo utilizando distintos conjuntos de datos de prueba que se han generado a partir de diferentes simuladores o usando casos reales. En la experimentación se han utilizado diferentes tipos de plataformas que se describen para cada caso concreto.

4.1 Introducción

Este capítulo analiza las prestaciones de los métodos descritos en el capítulo anterior. Este análisis se centra en evaluar experimentalmente la sensibilidad, especificidad y velocidad de los distintos métodos que se quieren comparar. La sensibilidad y la especificidad permiten evaluar la fiabilidad de los resultados, mientras que la velocidad, el tiempo necesario para obtenerlos.

Para la obtención de los resultados experimentales se han ejecutado los distintos alineadores con diversos bancos de pruebas en diferentes equipos informáticos. Entre los alineadores ejecutados se han incluido, además de los métodos propuestos en esta tesis, varios de los alineadores de ADN/ARN actualmente más utilizados. Permitiendo de esta forma comparar los resultados obtenidos por los distintos métodos propuestos entre sí y con los obtenidos por los alineadores de ADN/ARN más frecuentemente utilizados.

En el primer apartado de este capítulo se describe la configuración general de los experimentos, al igual que el software utilizado para generar los bancos de pruebas y la configuración por defecto de HPG Aligner.

En el siguiente apartado, se analiza el algoritmo HPG Aligner ARN BWT. Como se ha comentado anteriormente, se han desarrollado dos métodos diferentes de este algoritmo. El primero de ellos se basa en la asignación estática de tareas a hilos, y el segundo, en la asignación dinámica de tareas a hilos. En este apartado se comparan los resultados de estos dos métodos junto con los resultados obtenidos por TopHat 2.

El siguiente apartado se centra en el estudio del algoritmo HPG Aligner ARN BWT+M basado en la estructura metaexón. Este estudio pretende mostrar si la estructura metaexón mejora

los resultados obtenidos por las versiones anteriores del algoritmo. También se han comparado estos resultados con los obtenidos por algunos de los alineadores más usados para el alineamiento de secuencias de ARN.

Seguidamente, se analiza la implementación multinodo del algoritmo anterior, denominada HPG Aligner ARN BWT+M+MPI. El objetivo principal de este estudio es el de observar si el algoritmo escala adecuadamente y en cómo afecta esta nueva implementación a la sensibilidad de los resultados.

A continuación, se analizan los métodos basados en la matriz de sufijos para el procesamiento de secuencias de ADN/ARN, denominados HPG Aligner ADN SA y HPG Aligner ARN SA+M, respectivamente. Siguiendo la misma línea de los estudios anteriores, se ha evaluado la sensibilidad y velocidad de procesamiento de estos métodos.

El último apartado de este capítulo analiza los resultados obtenidos por el *framework* utilizando diferentes alineadores de ADN y de ARN.

Este análisis se centra en ver si el *framework* escala adecuadamente con el número de nodos utilizados y si la sensibilidad de los resultados se mantiene.

4.2 Configuración general de los experimentos

Para analizar las prestaciones de los distintos alineadores se han utilizado un conjunto de bancos de pruebas con diferentes características. Estos bancos se han generado utilizando simuladores y, en algunas ocasiones, casos de estudios reales. En concreto, se han utilizado los siguientes simuladores: dwgsim, simulador que forma parte del paquete SAMtools [26], y BEERS [16]. Ambos permiten generar bancos de pruebas con distintas configuraciones teniendo en cuenta, sobre todo, el total de lecturas que se quieren generar, su longitud, el porcentaje de inserciones y borrados, el porcentaje de errores o mutaciones y la cantidad de nucleótidos de la lectura que no han podido ser codificados (representados por el carácter «N»). En cada uno de los experimentos se detallan los valores concretos de estos parámetros utilizados para generar los correspondientes bancos de pruebas. Estos simuladores, además de generar las lecturas que se procesarán, también generan para cada lectura: su posición genómica, es decir, la hebra (positiva o negativa) en la que está; el número de cromosoma donde se encuentra, y su posición dentro del cromosoma. Además, el simulador BEERS, también proporciona el CIGAR del alineamiento, lo que permite obtener una mayor precisión en el análisis de la sensibilidad. Cuando se disponga del CIGAR, una lectura se considerará correctamente mapeada cuando las coordenadas de su alineamiento y su CIGAR coincidan con los especificados por el simulador.

Por otro lado, los sistemas donde se han llevado a cabo los experimentos han sido muy variados ya que dependen de las necesidades del algoritmo evaluado. Por esto, en cada experimento se describen las características del sistema en el que se ha llevado a cabo.

Por último, para todos los experimentos se han utilizado los siguientes valores por defecto de HPG Aligner: el tamaño mínimo de CAL a 20 nts; los parámetros de SWA, a sus valores por defecto (acierto: 5, error: -4, apertura hueco: 10, y extender hueco: 0,5); el tamaño mínimo de intrón a 40 nts; y el tamaño máximo de intrón a 500.000 nts.

4.3 Evaluación de HPG Aligner ARN BWT

En este apartado se muestran los resultados experimentales de HPG Aligner ARN BWT-S y de HPG Aligner ARN BWT-D. Como se vio en el capítulo anterior, ambas aplicaciones están diseñadas para ser ejecutadas en sistemas multiprocesador, están estructuradas en una serie de

etapas y explotan el paralelismo tanto dentro de cada etapa (lecturas de un paquete), como entre etapas (paquetes de lecturas). La principal diferencia entre ambos alineadores es la distribución de tareas entre los procesadores del sistema. En HPG Aligner ARN BWT-S se realiza un reparto estático, mientras que en HPG Aligner ARN BWT-D, se realiza un reparto dinámico que varía en función de la evolución de la carga de trabajo.

Los resultados experimentales presentados en este apartado se han obtenido en una plataforma equipada con 4 procesadores AMD Opteron 6276 a 2,3 GHz, con 16 núcleos cada uno (64 en total), y 64 GiB de memoria RAM. El banco de pruebas de prueba utilizado en los experimentos está formado por cuatro conjuntos de datos de entrada *single-end* con 2 millones de lecturas generadas por el simulador dwgsim. Estos conjuntos de datos se diferencian por el tamaño de lectura (100 o 250 nts) y el porcentaje de errores ($\varepsilon = 0{,}1\,\%$ o $1\,\%$), el 30 % de los cuales son inserciones/-borrados. Estos conjuntos se han referenciado de la siguiente forma: T1 para las lecturas de 100 nts y $\varepsilon = 0{,}1\,\%$, T2 para las lecturas de 100 nts y $\varepsilon = 1\,\%$, T3 para las lecturas de 250 nts y $\varepsilon = 0{,}1\,\%$ y T4 para las lecturas de 250 nts y $\varepsilon = 1\,\%$.

A continuación se presentan los experimentos realizados. El primero de ellos compara las prestaciones obtenidas por la versión estática de HPG Aligner ARN BWT-S, la versión dinámica HPG Aligner ARN BWT-D y TopHat 2 (+Bowtie 2); con los cuatro conjuntos de datos y con 4, 8, 16, 32 y 64 hilos/procesadores. El tamaño de las semillas (etapa C) se ha fijado a 16 nts en este primer experimento.

Para el estudio de HPG Aligner ARN BWT-S se ha evaluado el tiempo por separado de cada una de las etapas *B*–*E*, para un tamaño de lecturas de 100 nts, y se ha fijado una configuración de hilos por defecto para cada una de ellas. La configuración por defecto se ha particularizado para cada caso en concreto.

Las gráficas de la Figura 4.1 comparan el tiempo de ejecución de HPG Aligner ARN BWT-S frente a HPG Aligner ARN BWT-D, para los cuatro conjuntos de datos y 4, 8, 16, 32 y 64 hilos/procesadores. Los experimentos confirman que, cuando el número de hilos utilizado es de pequeño a moderado (4–16), hay una clara ventaja usando el reparto dinámico de hilos, pero la diferencia se difumina cuando el número de hilos crece. Por ejemplo, el tiempo de ejecución para el experimento T1 para 4, 8 y 16 hilos se ha reducido de 375,51 s, 180,62 s y 92,48 s, respectivamente, para HPG Aligner ARN BWT-S, a solo 175,13 s, 106,86 s y 65,32 s, para HPG Aligner ARN BWT-D. Esto corresponde a una aceleración de 2,14 , 1,69 y 1,41 para 4, 8 y 16 hilos, respectivamente. Sin embargo, este valor decrece rápidamente a 1,09 y 1,07 cuando el número de hilos aumenta a 32 y 64, respectivamente. Además, a medida que crece el número de hilos en la versión dinámica, el tiempo de ejecución aumenta considerablemente (de 65,32 s con 16 hilos a 73,54 s y 83,54 s con 32 y 64 hilos, respectivamente). Esto indica que no existe prácticamente margen de ganancia al incrementar el número de hilos más allá de 16 (al menos para estos bancos de pruebas). Los resultados obtenidos para el resto de conjuntos de datos son similares a los descritos para el conjunto T1.

Los dos últimos conjuntos de datos, T3 y T4, se han ejecutado utilizando varias configuraciones de hilos para la versión HPG Aligner ARN BWT-S. Por un lado, se han lanzado los experimentos para la configuración de hilos por defecto obtenida anteriormente y, por otro lado, se ha fijado una configuración optimizada para estos conjuntos de datos. Como puede observarse en la Figura 4.1, la configuración fijada para el tamaño de lectura de 100 nts no es óptima para las lecturas de 250 nts. Esto es debido a que cuando el tamaño de lectura crece, el tiempo que tarda cada una de las etapas es distinto al caso de 100 nts. Por este motivo, sería necesario realizar un nuevo estudio del tiempo de ejecución por etapa para fijar una configuración de hilos óptima. La mejora que se puede obtener cuando se realiza un estudio previo de la asignación óptima para un caso particular se ve reflejada en los resultados mostrados en las gráficas de la Figura 4.1 para los

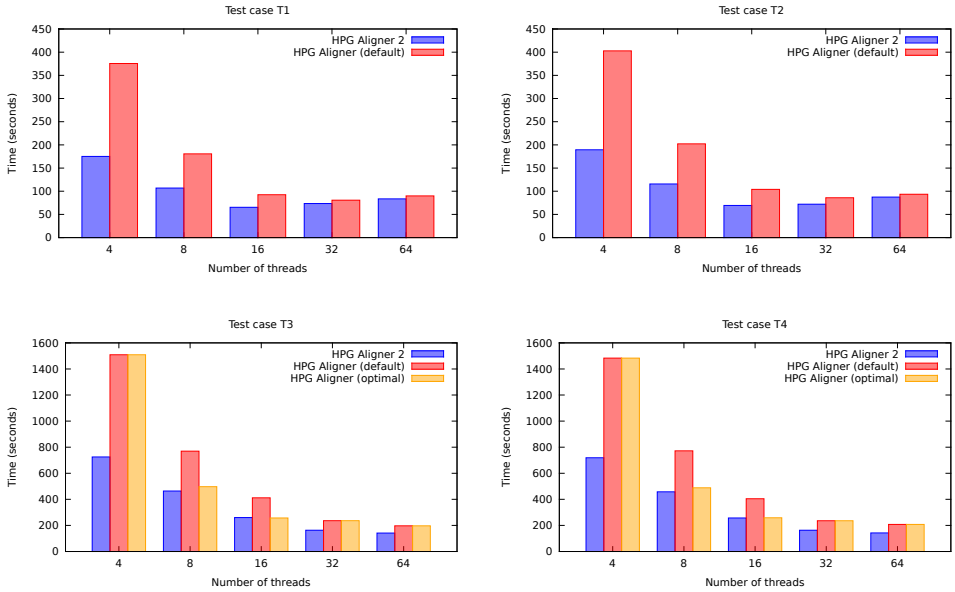


Figura 4.1: Tiempo de ejecución (en segundos) de la implementación estática frente a la dinámica, para los conjuntos de datos T1, T2, T3 y T4

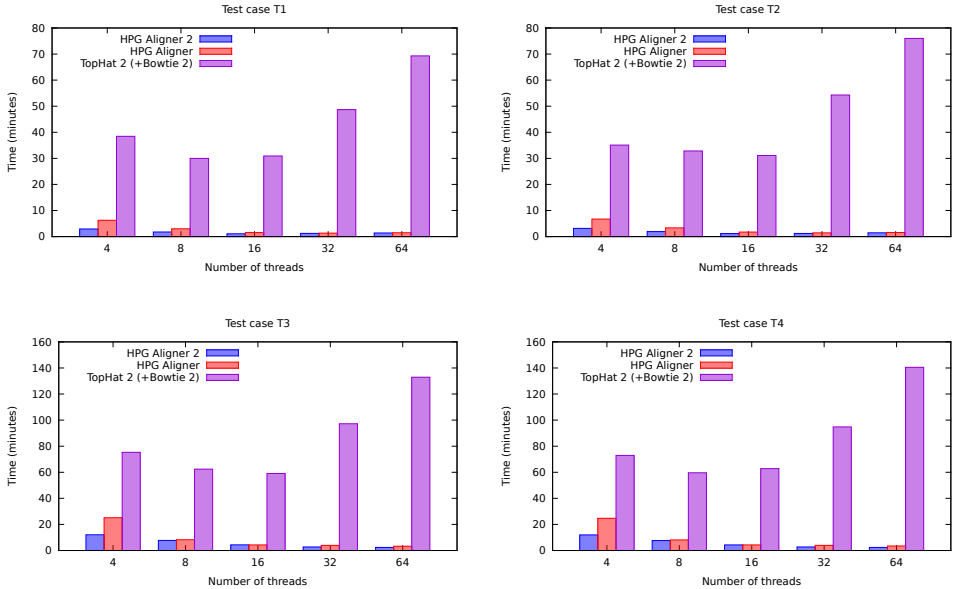


Figura 4.2: Tiempo de ejecución (en este caso en minutos) de la implementación dinámica frente a la estática y a TopHat 2 (+Bowtie 2) para los conjuntos de datos T1, T2, T3 y T4

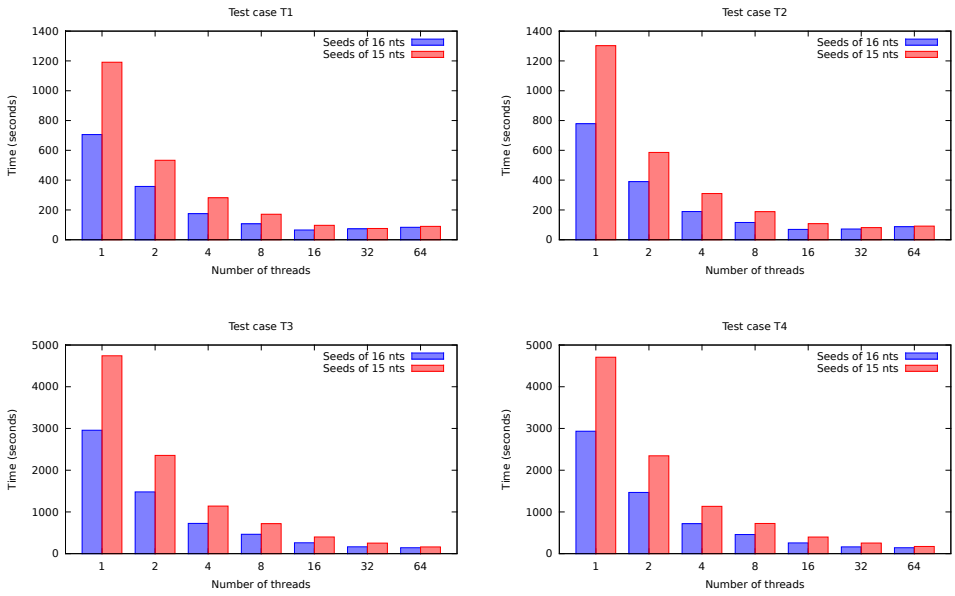


Figura 4.3: Tiempo de ejecución (en segundos) de la implementación HPG Aligner dinámica, con semillas de 16 y 15 nts, respectivamente

conjuntos de datos T3 y T4. Por otro lado, puesto que la versión de HPG Aligner ARN BWT-D realiza una asignación dinámica de hilos a etapas, este método no presenta la problemática anterior.

Las gráficas de la Figura 4.2 muestran la mejora de HPG Aligner ARN BWT-D frente a HPG Aligner ARN BWT-S configurado con una asignación óptima de hilos a etapas y a TopHat 2 (+Bowtie 2) con los cuatro bancos de pruebas y con 4, 8, 16, 32 y 64 hilos/procesadores. Estos resultados muestran claramente que ambas implementaciones de HPG Aligner ARN BWT superan a TopHat 2 (+Bowtie 2). Para los casos T1 y T2, HPG Aligner BWT ARN-D es aproximadamente 11 veces más rápido que TopHat 2 mientras, que para los casos T3 y T4, es alrededor de 5 veces más rápido.

El objetivo del siguiente grupo de experimentos se centra en analizar el impacto del tamaño de las semillas en la sensibilidad y en el rendimiento de HPG Aligner ARN BWT-D para los cuatro conjuntos de datos T1–T4 y 4–64 hilos. Este estudio viene motivado por los resultados mostrados en [38], donde se observa que la etapa C (búsqueda de regiones) es la más costosa, e influye de forma sustancial en las prestaciones. Puesto que el tiempo de ejecución de esta etapa se puede reducir aumentando el tamaño de las semillas, queda por ver cómo este aumento afectará a la sensibilidad del algoritmo. Los resultados mostrados en el Cuadro 4.1 muestran que el uso de semillas de tamaño 16 nts, en vez del tamaño original de 15 nts propuesto en [38], apenas afecta a la sensibilidad, pero si supone un decremento notable del tiempo de ejecución, como puede observarse en la Figura 4.3. Por ejemplo, para el caso T1, dividiendo las lecturas en fragmentos de 16 nts en vez de 15 nts, la relación obtenida va de 1,07 (64 hilos) hasta 1,68 (4 hilos). También hemos realizado experimentos con tamaños mayores de semilla, pero lamentablemente, al incrementar el tamaño de semillas más allá de 16 nts, la sensibilidad disminuye y el tiempo de ejecución aumenta. El Cuadro 4.1 muestra también la sensibilidad obtenida con TopHat 2. La sensibilidad de HPG Aligner ARN BWT-D (tamaño de semilla 16 nts) varía entre 96.4 % y 98.6 %, mientras que la obtenida por TopHat 2 ronda el 61.1 %. Los resultados tan extremadamente bajos de TopHat 2 (+Bowtie 2) T3 y T4 son

debidos a que TopHat 2 detiene el procesamiento de una lectura cuando este tarda demasiado tiempo.

Test	HPG Aligner Dinámico semillas de 15 nts	HPG Aligner Dinámico semillas de 16 nts	TopHat 2 (+Bowtie 2)
T1	97,8	97,5	61,1
T2	96,9	96,4	45,8
T3	98,5	98,6	10,0
T4	98,4	98,4	3,3

Cuadro 4.1: Sensibilidad (en %) de la implementación dinámica, con semillas de 15 y 16 nts, y de TopHat 2 (+Bowtie 2)

4.4 Evaluación de HPG Aligner ARN BWT+M

La evaluación del alineador HPG Aligner ARN BWT+M multihilo se centra principalmente en estudiar cómo afecta a la sensibilidad y al tiempo de ejecución, la utilización de la estructura metaexón. Esta estructura, como se describió anteriormente, se utiliza para facilitar el mapeo de las lecturas más conflictivas y acelerar el procesamiento del resto. Este estudio se ha realizado teniendo en cuenta las prestaciones obtenidas por los alineadores más usados y conocidos actualmente para el análisis del ARN, que son: TopHat(v2.0.9) [50], STAR (v2.3.0) [12], Mapsplice (v2.1.3) [54] y HISAT [22].

Para este análisis se han utilizados tres tipos de bancos de pruebas distintos. Dos de ellos se han obtenido mediante los simuladores BEERS [16] y dwgsim 0.18 [26]. El tercero se ha obtenido a partir de datos reales descargados del archivo SRA (*Sequence Read Archive*).

Mediante el simulador BEERS se han generado 18 bancos de pruebas de 10 millones de lecturas cada uno, que difieren en el tamaño de las lecturas y en el porcentaje de error. Las longitudes y errores fijados son: 50, 75, 100, 150, 250 y 400 nts; y 0,1 , 1 y en 2 %; respectivamente. Para todas las configuraciones, el ratio de inserciones y borrados se ha fijado al 10 %. Cabe destacar que todas las lecturas se han generado a partir de 10.000 genes del genoma humano.

La aplicación dwgsim ha sido ejecutada en el modo «Illumina» mediante la opción «-c 0» para los mismos seis tamaños de lecturas que el caso anterior. Para estudiar la sensibilidad frente a las mutaciones y a las lecturas que contienen nucleótidos no codificados «N», se han generado 3 conjuntos de datos diferentes. El primero se ha generado para un ratio de error de 0,1 % y un máximo de 2 Ns por lectura. Para el segundo y el tercer conjunto de datos se incrementó el ratio de errores a 1 % y 2 %, respectivamente, con 3 Ns por lectura. En ambas configuraciones, el ratio de inserciones y borrados se ha fijado a 10 %. Las lecturas han sido generadas a partir de la versión Ensembl 68 GRCh37 del genoma humano. Para el conjunto de datos «paired-end» la distancia entre las parejas se ha fijado a 400 nts para lecturas inferiores a 250 nts y a 250 nts para lecturas entre 250 y 400 nts.

Los bancos de pruebas basados en datos reales son dos ficheros descargados del archivo SRA. El primero es el fichero SRR364003, que contiene lecturas generadas por el secuenciador Illumina HiSeq 2000 y está formado por 81,6 millones de lecturas «single-end» y «paired-end» de 100 nts; el segundo es el fichero SRX025091, que contiene lecturas generadas por el secuenciador Roche 454 GS FLX y está formado por 1,26 millones de lecturas «single-end» de aproximadamente

600 nts. Se han escogido estos dos ficheros por sus características, ya que el primero de ellos contiene un gran número de lecturas y de esta forma se podrá estudiar con detalle el rendimiento de los alineadores. El segundo fichero se ha seleccionado por el tamaño de sus lecturas, ya que están formadas por 600 nts y esto permitirá estudiar como se comportan los alineadores frente a estos tamaños de lecturas tan elevados.

Los experimentos se han realizado en un servidor equipado con 2 hexa-core Intel Xeon E5645 a 2,40 GHz y con 48 GiB de memoria RAM. En todas las ejecuciones se ha fijado el número de hilos a 12.

Los secuenciadores actuales están incrementado progresivamente el tamaño de las lecturas generadas: los HiSeq 1500 o HiSeq 2500 producen lecturas de 150 nts, le Roche 454 GS FLX+ genera lecturas de 700 −800 nts y el Pacific Biosciences de hasta 3.000 nts. Por lo tanto, es importante que los alineadores mantengan el ratio de lecturas correctamente mapeadas conforme el tamaño de las lecturas aumenta.

El Cuadro 4.2 muestra las prestaciones obtenidas por HPG Aligner BWT+M y por el resto de alineadores evaluados con los bancos de pruebas generados por el simulador BEERS. La primera columna del cuadro muestra el tamaño de las lecturas; la segunda, el ratio de error de las lecturas; y las siguientes columnas muestran para cada uno de los alineadores seleccionados: el ratio de lecturas no mapeadas (RNM), el ratio de lecturas correctamente mapeadas (CMR) y el tiempo de ejecución (en minutos). Como se puede ver en el cuadro, para todos los tamaños de lecturas y una tasa de error del 0,1 %, todos los alineadores proporcionan una sensibilidad muy similar. Sin embargo, a medida que aumenta el tamaño de las lecturas y el porcentaje de errores, la sensibilidad obtenida por HPG Aligner BWT+M aumenta considerablemente en relación a la del resto de alineadores. Para lecturas de tamaño superior a 250 y a 400 nts, la sensibilidad de HPG Aligner ARN BWT+M y STAR con respecto a TopHat 2 y MapSplice 2 crece considerablemente. También conviene destacar que HPG Aligner ARN BWT+M, STAR y MapSplice 2 mantienen el porcentaje de lecturas no mapeadas por debajo del 5 % para todos los tamaños de lecturas.

En cuanto al tiempo de ejecución, HPG Aligner ARN BWT+M es el alineador más rápido, obteniendo aceleraciones desde 7x hasta 10x para lecturas de 50 y 150 nts, respectivamente, con respecto a TopHat 2; de 7x con respecto a MapSplice 2; y de 2x con respecto a STAR. El tiempo obtenido por HISAT es muy similar al de HPG Aligner ARN BWT+M, incluso llega a ser ligeramente inferior cuando se procesan lecturas muy pequeñas (aunque esto en realidad no supone una gran ventaja, puesto que como se ha comentado, las lecturas generadas por los secuenciadores tienden a ser cada vez más grandes).

Estos experimentos se han repetido para los bancos de pruebas generados con el simulador dwgsim. Los resultados obtenidos son muy similares a los mostrados y por eso no se han incluido en este apartado (pueden consultarse en el material complementario de [39]). De forma general se puede concluir que a medida que aumenta el tamaño de las lecturas y la tasa de errores, la sensibilidad de todos los alineadores decrece. En particular, la sensibilidad de TopHat 2 decrece notablemente para el porcentaje de error del 0,1 %. Uno de los motivos por los cuales la sensibilidad es inferior en el caso de los bancos de pruebas generados con dwgsim, es porque este simular los errores producidos por los propios secuenciadores, por lo que el número de mutaciones es ligeramente superior al de los generados con BEERS.

El Cuadro 4.3 muestra las prestaciones obtenidas por los cinco alineadores evaluados tras el procesamiento de los bancos de pruebas con datos reales. La primera columna del cuadro muestra el tamaño de las lecturas; la segunda columna, la cantidad de lecturas en millones y si el conjunto de datos se trata de «single-end» (se) o «paired-end» (pe); y las siguientes columnas muestran para cada uno de los alineadores seleccionados: el ratio de lecturas no mapeadas (RNM), el ratio de lecturas correctamente mapeadas (CMR) y el tiempo de ejecución (en minutos). En este cuadro

RL	MR	HPG Aligner			HISAT			STAR			TopHat 2+Bowtie 2			MapSplice		
(nts)	(%)	RNM	CMR	T	RNM	CMR	T	RNM	CMR	T	RNM	CMR	T	RNM	CMR	T
	0,1	0,42	96,50	4,80	3,22	92,36	4,45	1,46	91,22	8,28	0,93	97,02	36,35	1,16	96,02	26,93
50	1	0,84	94,90	4,90	8,61	81,60	4,80	1,72	87,05	8,73	3,04	94,90	79,65	2,03	95,02	26,13
	2	2,26	91,92	5,20	17,18	66,53	4,58	3,16	81,40	8,85	10,18	87,82	374,90	4,81	91,98	26,08
	0,1	0,24	96,25	5,80	1,13	96,57	6,21	0,46	91,73	9,52	2,98	96,74	48,22	0,24	96,45	37,32
75	1	0,94	94,89	6,00	3,48	91,88	6,40	1,18	87,00	9,97	6,90	91,86	50,78	1,16	95,25	35,05
	2	1,27	93,09	6,40	10,46	78,95	6,46	1,55	81,83	11,25	22,18	76,88	58,93	2,20	92,67	34,28
	0,1	0,24	95,69	7,40	1,24	96,19	7,86	0,64	91,77	11,77	1,94	96,37	65,08	0,26	96,20	51,17
100	1	0,55	94,50	7,50	4,51	89,79	8,15	0,62	87,48	12,08	11,70	87,31	74,02	0,68	95,34	46,03
	2	0,93	93,00	7,80	17,19	67,17	8,40	1,14	82,38	12,35	36,14	63,26	85,65	1,50	92,05	45,60
	0,1	0,31	94,11	10,50	1,56	95,29	11,75	0,57	91,03	15,07	3,55	94,96	97,90	0,21	94,34	56,88
150	1	1,95	90,28	10,60	13,03	74,48	11,41	2,73	84,39	15,08	27,13	72,05	121,55	2,67	89,10	53,53
	2	0,78	92,16	10,90	39,76	35,82	11,91	0,73	83,30	15,73	61,84	37,71	140,28	1,57	84,78	56,73
	0,1	0,66	91,13	16,55	2,47	93,04	17,80	0,62	90,25	21,58	6,70	91,68	185,78	0,23	88,26	75,55
250	1	0,81	90,76	16,50	31,60	46,12	19,28	0,71	86,30	22,30	53,29	46,43	256,83	0,75	69,40	79,28
	2	1,62	89,66	17,45	77,04	5,21	17,46	1,43	80,22	23,35	89,62	10,31	309,85	1,24	63,49	76,37
	0,1	1,08	88,90	25,45	5,04	88,08	28,50	0,56	89,46	32,12	12,32	86,69	369,58	0,03	82,64	106,67
400	1	3,35	83,71	25,60	66,17	11,31	29,70	4,26	80,92	33,83	82,40	17,51	573,18	1,65	46,71	103,98
	2	2,84	85,38	26,00	96,82	0,10	27,10	2,75	64,50	24,50	99,02	0,97	603,28	0,51	45,15	106,53

Cuadro 4.2: Prestaciones de los distintos alineadores con los bancos de pruebas generados con BEERS

se puede observar que para el fichero SRR364003, que contiene 81 millones de lecturas de 100 nts, HPG Aligner ARN BWT+M ha sido capaz de mapear el 80 % de las lecturas, en cambio HISAT y MapSplice 2 se quedan en un 76 %, STAR en un 72 % y TopHat 2 en un 63 %. En términos de tiempo de ejecución, HPG Aligner ARN BWT+M, HISAT y STAR tienen aproximadamente unos tiempos 10 veces inferiores a los de TopHat 2 y MapSplice 2, los cuales rondan los 100 minutos de ejecución. Para el fichero SRP003173, que contiene lecturas de aproximadamente 600 nts, HPG Aligner ARN BWT+M mapea alrededor de un 50 %, mientras que STAR un 30 %, HISAT y TopHat 2 tienen un porcentaje de mapeo muy pequeño y, finalmente, MapSplice 2 no ha sido capaz de generar ningún resultado tras 3 días de ejecución.

Por otro lado, además de qué tamaño de lectura soportan adecuadamente los distintos alineadores, también se debe tener en consideración cómo aumenta su tiempo de ejecución conforme se incrementa el volumen de datos a procesar. En el Cuadro 4.4, donde se recoge este análisis, se puede observar como HPG Aligner ARN BWT+M y HISAT proporcionan un tiempo de procesamiento mucho menor que el del resto de alineadores.

Para finalizar este apartado, se analiza por último el comportamiento de los alineadores en función de si las lecturas contienen o no punto de unión entre exones. Para ello, se han generado dos conjuntos de datos con 5 millones de lecturas de 100 nts cada uno. En uno de estos conjuntos, todas las lecturas contienen puntos de unión, mientras que en el otro, ninguna de sus lecturas contienen puntos de unión. En contra de lo que cabría esperar, los tiempos los tiempos de ejecución obtenidos con ambos conjuntos son muy similares (220.61 segundos sin uniones y 222,45 s con uniones). Tras repetir los experimentos para lecturas simuladas de 250 nt se obtuvieron resultados similares (485,56 s para las lecturas sin uniones y 488,74 s para las lecturas con uniones). En [39] pueden consultarse los detalles de estos resultados experimentales.

RL	Tamaño	HPG Aligner			HISAT			STAR			TopHat 2+Bowtie 2			MapSplice		
(nt)		RNM	CMR	Time	RNM	CMR	Time	RNM	CMR	Time	RNM	CMR	Time	RNM	CMR	Time
100	81M (se)	4,53	79,72	11,00	15,16	76,10	10,43	5,50	75,00	13,90	25,73	63,38	115,90	4,63	76,83	139,90
100	81M (pe)	4,90	78,20	21,60	15,00	77,80	21,46	8,00	72,02	22,70	29,26	59,90	230,60	5,04	77,50	288,10
$\tilde{6}00$	1,26M (se)	11,40	48,04	6,90	99,87	0,10	3,45	41,60	30,25	6,60	99,97	0,02	520,60	NA	NA	NA

Cuadro 4.3: Prestaciones de los distintos alineadores con los bancos de pruebas de datos reales

Millones lecturas	HPG Aligner	HISAT	STAR	TopHat2+Bowtie2	MapSplice
2	0,90	0,95	4,50	38,70	11,80
5	1,90	1,70	5,60	40,50	18,65
10	3,80	4,20	7,75	75,20	36,40
20	7,60	7,28	13,50	112,20	68,50

Cuadro 4.4: Tiempo de ejecución para generar un fichero BAM de los alineadores. Tiempo requerido (en minutos) por los distintos alineadores para generar un fichero BAM a partir de ficheros de entrada con distintos números de lecturas

4.5 Evaluación de HPG Aligner ARN BWT+M+MPI

En este apartado se evalúan las prestaciones de HPG Aligner ARN BWT+M+MPI: la versión multinodo del alineador HPG Aligner ARN BWT+M. Este análisis se centra en el estudio de cómo puede acelerarse la ejecución de esta aplicación utilizando más nodos de cómputo. Para ello, se ha utilizado un clúster compuesto por 12 nodos, cada uno con un procesador Intel Xeon E645 hexa-core a 2,40 GHz: 144 procesadores en total. Cada nodo tiene 24 GiB de memoria RAM. Los nodos están interconectados por una red Infiniband QDR (por medio de un conmutador Mellanox MTS3600).

Los bancos de pruebas utilizados son dos ficheros «single-end», que contienen 5 y 80 millones de lecturas de 100 nts, respectivamente, generados con el simulador BEERS [16]. El ratio de errores seleccionado fue del 0,1 %, y la frecuencia de inserciones y borrados se fijó al valor por defecto (0.5 %).

El Cuadro 4.5 muestra el tiempo de ejecución obtenido por HPG Aligner BWT+M+MPI para el procesamiento del fichero de entrada de 80 millones de lecturas usando $p = 1 \ldots 12$ nodos y 12 hilos por nodo. En el cuadro también se muestra la aceleración de la versión paralela frente a la secuencial. Estos resultados muestran que el tiempo se reduce de 12 minutos, para la versión multihilo ejecutada en un nodo, a solo 1,5 minutos para la versión HPG Aligner BWT+M+MPI con 12 nodos. En términos de escalabilidad, la aplicación consigue un paralelismo notable hasta los 6 nodos, y un paralelismo más suave cuando se emplea un mayor número de nodos.

El Cuadro 4.6 muestra el tiempo de ejecución de cada uno de los tres flujos de trabajo, además del tiempo de comunicación necesario para realizar la unión de los resultados de los distintos nodos tras cada flujo de trabajo. El cuadro muestra la media y desviación estándar del tiempo obtenido al realizar 5 ejecuciones. Como puede observarse, la desviación es muy pequeña, lo que indica que la carga de trabajo asignada a cada nodo está equilibrada. Además, el coste de la fusión de los resultados de los distintos nodos varía entre 0.57 y 3.68 segundos, siendo despreciable comparado con el tiempo de ejecución total de los tres flujos de trabajo.

	Nodos (p)						
	1	2	4	6	8	10	12
Tiempo total	733,00	387,93	201,95	139,92	118,81	103,56	92,07
Aceleración	1,00	1,89	3,63	5,24	6,17	7,08	7,96

Cuadro 4.5: Tiempo de ejecución (en segundos) y aceleración de HPG Aligner ARN BWT+M+MPI con el conjunto de datos de 80 millones de lecturas de 100 nts

	Nodos (p)						
	1	2	4	6	8	10	12
Flujo 1	688,05	351,90±3,35	173,98±2,89	117,12±0,76	94,23±3,76	79,81±4,11	66,65±4,28
Unión 1		1,52	2,32	3,06	2,94	3,68	3,54
Flujo 2	41,38	20,98±0,24	10,67±0,11	7,28±0,04	5,59±0,10	4,65±0,20	3,83±0,05
Unión 2		1,12	1,64	2,15	2,10	2,58	2,54
Flujo 3	3,57	2,01±0,09	1,34±0,21	1,05±0,21	0,83±0,15	0,72±0,14	0,65±0,14
Unión 3		0,57	1,11	1,62	1,59	2,09	2,06

Cuadro 4.6: Tiempo medio de ejecución (en segundos) de cada uno de los tres flujos de trabajo de HPG Aligner ARN BWT+M+MPI y de la unión de los resultados intermedios con el conjunto de datos de 80 millones de lecturas de 100 nts

El Cuadro 4.7 muestra los efectos que tiene sobre la sensibilidad del algoritmo la fusión, de las estructuras metaexón y AVL. El banco de pruebas es en este caso un conjunto de datos formado por 5 millones de lecturas. Dentro de este conjunto de datos, se han seleccionado las lecturas en función del número de exones que estas abarcan. Como puede observarse en el Cuadro 4.7, la primera columna indica el número de exones de cada uno de los grupos y la segunda, que porcentaje de todas lecturas forman parte de ese grupo. Estos resultados muestran que la ejecución de la versión multihilo en un nodo obtiene una sensibilidad del 95.64 %, mientras que la versión MPI en 12 nodos cuando se unen los resultados de todos los nodos tras cada flujo de trabajo obtiene un valor un poco más reducido, del 95.27 %; esto es, una diferencia mínima: del 0.37 %. Esta diferencia es ligeramente mayor cuando no se unen los resultados de los distintos nodos. También conviene señalar que la sensibilidad es ligeramente superior en el caso de la versión MPI 2 nodos que en el de la versión multihilo en un nodo. Esta diferencia se explica por las pequeñas variaciones de sensibilidad que pueden darse dependiendo del orden en el que se procesen las lecturas.

4.6 Evaluación de HPG Aligner ADN SA+M

Como se ha comentado anteriormente, las aplicaciones más utilizadas para el alineamiento de secuencias de ADN son Bowtie 2 y BWA MEM. Así pues, para poner en contexto las prestaciones de HPG Aligner ADN SA, también se han evaluado las prestaciones de estos dos alineadores. En concreto, se han ejecutado las versiones 2.2.1 de Bowtie y 0.7.5a de BWA MEM. El sistema donde se han lanzado los experimentos está formado por 2 CPU Intel Xeon E5645 2,40 GHz CPUs (lo que supone un total de 12 procesadores) y 48 GiB de memoria RAM. En todas las ejecuciones se han utilizado 12 hilos. Los bancos de pruebas son de dos tipos: simulados, mediante la aplicación dwgsim 0.1.10; y reales, obtenidos a partir del genoma Drosophila obtenido de PacBio.

			Nodos (p)						
Exones	Lecturas	Configuración	1	2	4	6	8	10	12
1	75,44	Con unión	98,82	98,82	98,82	98,82	98,82	98,82	98,82
		Sin unión		98,82	98,82	98,82	98,82	98,82	98,82
2	22,34	Con unión	87,62	88,03	87,67	87,28	87,03	86,75	86,54
		Sin unión		87,46	86,78	85,97	85,35	84,71	84,12
3	2,16	Con unión	68,56	69,78	67,97	66,34	64,91	63,97	63,04
		Sin unión		68,03	65,79	63,48	61,37	59,92	58,24
>3	0,06	Con unión	45,48	48,12	44,62	40,98	39,19	36,76	35,58
		Sin unión		43,23	40,41	37,98	35,16	33,01	32,08
>=1	100,00	Con unión	95,64	95,76	95,63	95,51	95,42	95,34	95,27
		Sin unión		95,59	95,38	95,15	94,97	94,79	94,62

Cuadro 4.7: Sensibilidad de HPG Aligner ARN BWT+M+MPI con el conjunto de datos de 5 millones de lecturas de 100 nts

Los conjuntos de datos de prueba simulados se han generado en formato *single-end* a partir del genoma humano (versión Ensembl73, GRCh37). La aplicación dwgsim se ha ejecutado en el modo Illumina para generar 40 millones de lecturas de 100, 150, 400, 800, 2000 y 5.000 nts. Además, se ha generado un conjunto de datos de alta calidad conteniendo únicamente un 0,1 % de mutaciones y un segundo conjunto de datos de peor calidad con un 1 % de mutaciones. En ambos casos, el 10 % de mutaciones son inserciones y borrados y, además, el 30 % de estos errores se han extendido para que estén formados por más de una inserción o borrado. Finalmente, el máximo número de N ha sido fijado a 2.

El Cuadro 4.8 muestra los resultados obtenidos por HPG Aligner ADN SA, BWA MEM y Bowtie 2. Las lecturas son consideradas correctamente mapeadas si el cromosoma, la hebra y la posición de mapeo (con un margen de error de +/-5 nts) coinciden con las coordenadas del alineamiento indicado por dwgsim. Como puede observarse, mientras los porcentajes de mapeos de lecturas correctos son similares, el tiempo de ejecución de HPG Aligner ADN SA es significantemente inferior al obtenido por BWA MEM, especialmente, cuando el tamaño de las lecturas aumenta, llegando hasta un 18x para tamaños de 5.000 nts. Por otro lado, Bowtie 2, obtiene un tiempo de procesamiento mucho mayor que HPG Aligner ADN SA y que BWA MEM, además de ser incapaz de terminar el procesamiento cuando el tamaño de las lecturas es superior a 800 nts.

Adicionalmente, se ha estudiado el efecto de las inserciones y borrados sobre un conjunto de datos que contienen huecos mucho mayores en sus lecturas (tamaño mínimo de hueco: 5, 7, 10 y 20 nts) para tamaños de lecturas de 100, 150, 400 y 800 nts (en el material suplementario de [48] se recogen los cuadros con los resultados experimentales). En el escenario más complicado (lecturas de 800 nts con huecos >20 nts), el tiempo de ejecución de HPG Aligner ADN SA es comparable al de BWA MEM, en cambio, la sensibilidad de HPG Aligner ADN SA es notoriamente superior (80,12 % frente 63,73 %). En cuanto a Bowtie 2, su sensibilidad es generalmente inferior.

Para el conjunto de datos reales que contiene 1 millón de lecturas gran longitud, se ha constatado que los alineadores a menudo generan alineamientos no muy realistas. Estos alineamientos contienen unas pocas decenas de nucleótidos mapeados, mientras que todos los demás son inserciones. Para evitar que estos alineamientos puntúen como correctos, se ha añadido el requisito

Tamaño	Mutaciones	HPG Aligner		BWA MEM		Bowtie 2	
lecturas (nts)	(%)	CM	Tiempo	CM	Tiempo	CM	Tiempo
100	0,1	98,77	20,57	96,99	29,34	94,67	29,40
	1	98,22	19,66	96,65	33,34	92,98	29,15
	2	97,45	17,46	96,11	37,62	90,52	29,04
150	0,1	99,54	22,90	98,09	43,35	96,71	47,61
	1	99,29	22,09	97,96	49,12	95,93	46,50
	2	98,96	18,13	97,72	54,03	94,73	46,36
400	0,1	99,93	31,35	99,12	124,16	98,82	209,26
	1	99,78	30,49	99,06	142,81	98,71	221,92
	2	99,58	26,30	98,95	157,65	98,56	200,11
800	0,1	99,95	35,57	99,42	279,54	99,29	4604,90
	1	99,74	35,00	99,38	312,55	99,24	2750,26
	2	99,47	34,70	99,28	340,46	99,18	2894,38

Cuadro 4.8: Prestaciones de HPG Aligner ADN SA, BWA MEM y Bowtie 2

de que el alineamiento obtenido debe cubrir al menos un 80 % de la lectura para poder ser considerado correcto. HPG Aligner ADN SA obtiene un porcentaje de mapeo del 93.21 %, que después de eliminar las lecturas que no llegan al mínimo de cobertura queda en un 92.95 % de lecturas correctamente mapeadas. BWA MEM inicialmente mapea un 99.95 % de las lecturas. En cambio, cuando se excluyen las lecturas que no cubren el mínimo, se queda solamente en un 90.22 %. Además, mientras que HPG Aligner ADN SA completa el procesamiento en solo 27.51 min, BWA MEM requiere 130,34 min. Esto implica que se obtiene una aceleración de un 5x. Bowtie 2 no consigue finalizar el procesamiento del conjunto de datos debido a un fallo de memoria (*signal 9 kill*). BLASR [6] sí consigue obtener un buen porcentaje de mapeo correcto de lecturas, un 99.81 % pero el tiempo de ejecución es de 342 min.

Para evaluar el desempeño con lecturas pequeñas, se ha usado un conjunto de datos de 32,7 millones de lecturas de 100 nts procedentes de 1.000 genomas. Para este conjunto de datos se obtiene una aceleración de 1.5x con HPG Aligner ADN SA (14 min), con respecto a BWA MEM (21 min), y un porcentaje de alineamientos correcto del 96,30 % con HPG Aligner ADN SA, frente a un 97.13 % con BWA MEM.

4.7 Evaluación de HPG Aligner ARN SA+M

En este apartado se analizan las prestaciones de HPG Aligner ARN SA+M en su implementación multihilo. Este estudio tiene dos partes. En primer lugar, se hace un análisis del rendimiento y la escalabilidad del algoritmo. En segundo lugar, se estudia su sensibilidad y velocidad de procesamiento frente a las de otros alineadores muy utilizados.

La evaluación se ha realizado usando un banco de pruebas «single-end», generado con el simulador BEERS, que consta de 80 millones de lecturas del genoma humano con tamaños de 100 nt, 150 nt, 250 nt y 400 nt. Para cada uno de estos casos, se han configurado tres escenarios diferentes correspondientes a distintos ratios de errores, 0.1 %, 1 % y 2 %. La frecuencia de las inserciones y borrados se ha fijado al valor por defecto (0.05 %).

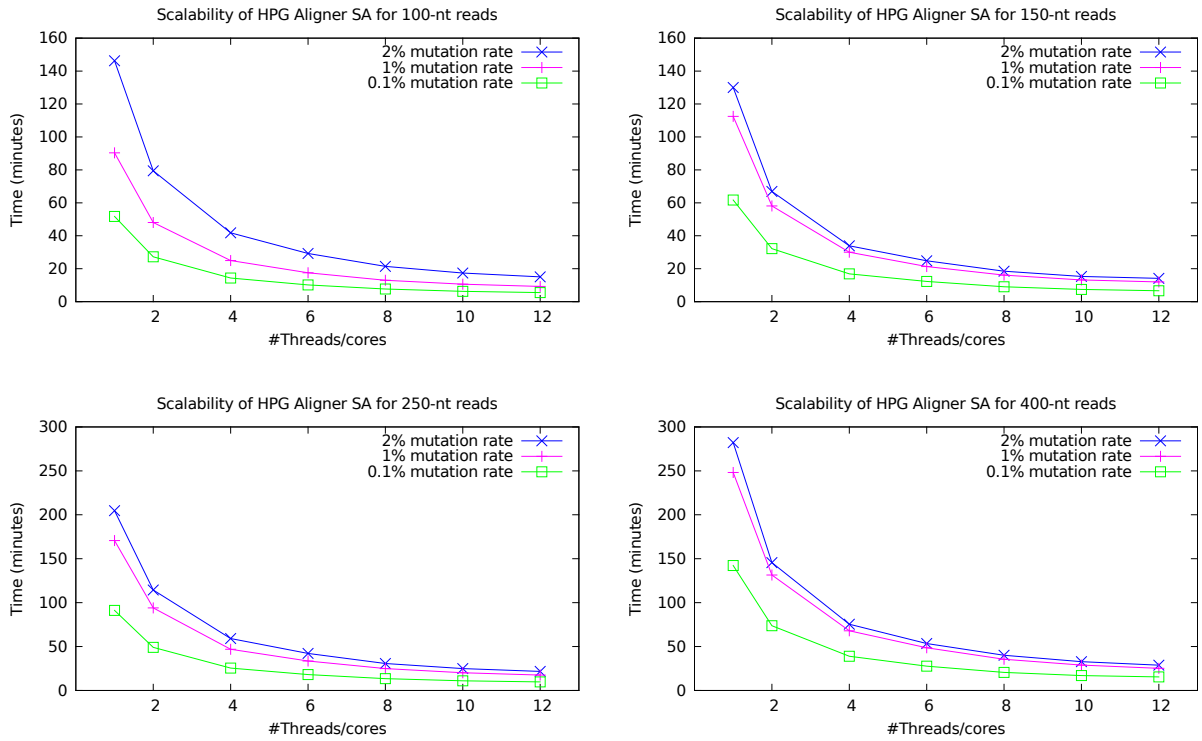


Figura 4.4: Rendimiento de la paralelización (tiempo de ejecución en minutos) de HPG Aligner SA+M con distintos tamaños de lecturas y ratios de mutaciones

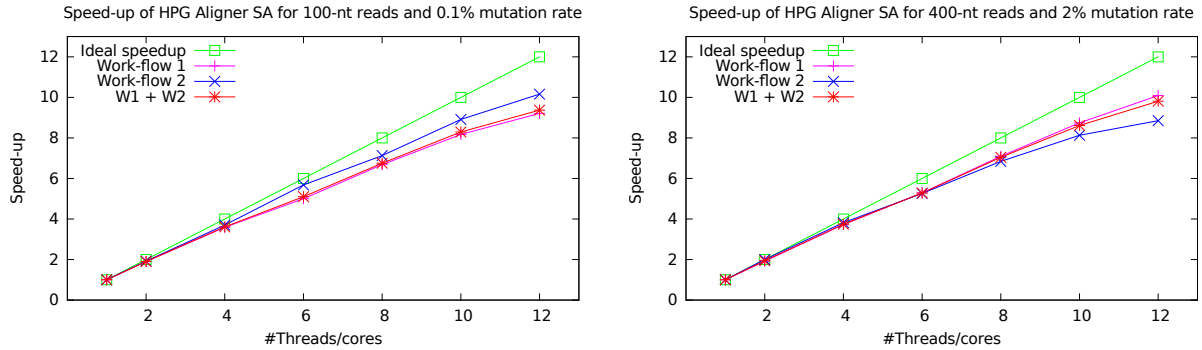


Figura 4.5: Aceleración de HPG Aligner SA+M con lecturas de 100 nts y ratio de mutación 0.1 % (izquierda); y con lecturas de 400 nts y ratio de mutación 2 % (derecha)

Tamaño	Ratio	Numero de hilos/Procesadores					
lecturas (nts)	mutaciones (%)	2	4	6	8	10	12
100	0,1	1,90	3,61	5,10	6,75	8,29	9,37
	1	1,88	3,62	5,15	6,96	8,51	9,75
	2	1,84	3,50	4,99	6,83	8,42	9,71
150	0,1	1,92	3,66	5,02	6,81	8,28	9,32
	1	1,94	3,75	5,27	6,99	8,54	9,40
	2	1,94	3,84	5,24	7,03	8,48	9,19
250	0,1	1,86	3,57	5,04	6,77	8,26	9,35
	1	1,82	3,64	5,09	6,86	8,44	9,73
	2	1,79	3,46	4,85	6,66	8,22	9,45
400	0,1	1,93	3,66	5,14	6,93	8,37	9,23
	1	1,89	3,65	5,11	6,98	8,59	9,85
	2	1,94	3,74	5,28	7,04	8,61	9,81

Cuadro 4.9: Aceleración de HPG Aligner ARN SA+M con distintos tamaños de lecturas y ratios de mutaciones

Tamaño	Ratio	Flujo de trabajo 1					Flujo de trabajo 2		
lecturas (nts)	mutaciones (%)	A	B	C	D	E	A	B	C
100	0,1	2,14	6,19	18,41	11,59	4,60	0,20	7,77	0,43
	1	1,99	5,25	38,62	20,40	3,56	0,57	15,76	1,39
	2	1,92	5,02	59,28	28,45	2,97	0,92	42,44	2,30

Cuadro 4.10: Tiempo de ejecución (en minutos) de las etapas del flujo de trabajo 1 y 2 con lecturas de 100 nts y distintos ratios de mutaciones

Todos los experimentos han sido realizados en una plataforma equipada con dos procesadores Intel Xeon E5645 (6 núcleos por procesador) a 2,4 GHz, y con 48 GiB de memoria RAM. En todos los casos, cada hilo se ha asociado a un procesador diferente.

La Figura 4.4 muestra el tiempo de ejecución total de HPG Aligner ARN SA+M para 1, 4, 6, 8 y 12 hilos. Estos resultados muestran la dependencia entre el ratio de mutaciones/tamaño lectura y el tiempo de ejecución. El principal aspecto a resaltar es la significativa variación del tiempo de ejecución cuando el ratio de mutaciones se incrementa del 1 % al 2 % con lecturas de 100 nts (parte superior izquierda de la gráfica). Esta variación deja de ser tan destacable para el resto de tamaños (150, 250 y 400 nts). Concretamente, en los experimentos con un solo hilo, se observa que el tiempo de ejecución se incrementa en un factor de 1.62. En cambio, para el caso de 150 nts, este factor ya es considerablemente inferior; 1.15. Por otro lado, cuando el ratio de mutaciones se incrementa de 0,1 a 1 %, el comportamiento para los cuatro tipos de tamaño de lecturas es similar, con variaciones que van desde un factor de 1,63 (lecturas de 400 nts, 12 hilos) a 1.91 (250 nts, 2 hilos). Desde el punto de vista del tamaño de lectura, existe una relación muy cercana entre el tiempo de ejecución y los ratios de mutaciones 0.1 % y 1 %, mostrándose en ambos casos factores de 1,2 , 1,8 y 2,7 cuando el tamaño varía de 100 a 150, a 250 y a 400 nts, respectivamente. Cuando el ratio de

HPG Aligner	hpg-aligner rna -i /GENOMES/SA_INDEX/ -f simulate_dataset.fq –cpu-threads t -o outputHPG –fast-mode –read-batch-size 20000
STAR	STAR –genomeDir STAR_INDEX/ –readFilesIn simulate_dataset.fq –runThreadN t –outFileNamePrefix outputSTAR/
MapSplice	mapsplice.py -c MAP_SPLICE/ -x MAP_SPLICE/hs.73 -1 simulate_dataset.fq -p t -o outputMapSplice/
TopHat 2+Bowtie 2	tophat -p t –no-sort-bam –no-convert-bam -o outputTopHat2 Bowtie2_index/Homo_Sapiens_Bowtie simulate_dataset.fq

Cuadro 4.11: Comandos usados para la ejecución de los alineadores. En cada experimento, t, se ha reemplazado por un número concreto de hilos/núcleos

mutación es del 2 %, estas magnitudes varían notablemente, con factores alrededor del 0,9 , 1,4 y 1,9 cuando se varía de 100 a 150, a 250 y a 400 nts, respectivamente.

El Cuadro 4.9 muestra la aceleración de la ejecución paralela desde 1 hasta 12 hilos con respecto a la ejecución secuencial de HPG Aligner ARN SA+M. Desde el punto de vista de concurrencia, estos resultados revelan un paralelismo notable de HPG Aligner ARN SA+M cuando se escala hasta 12 núcleos (el mayor número posible en esta plataforma), con valores que son ligeramente independientes del ratio de mutaciones y del tamaño de lectura. Concretamente, con 12 hilos se obtiene una aceleración alrededor del de 9.5 para todos los casos. Además, la Figura 4.5 muestra la escalabilidad del tiempo de ejecución completo de HPG Aligner SA+M para los dos casos anteriores (100 nts, ratio mutaciones 0.1 %; 400 nts, ratio mutaciones 2 %). Estas gráficas muestran que el rendimiento del alineador (etiquetado como W1+W2) se acerca considerablemente al obtenido por el flujo de trabajo 1, mientras que el flujo de trabajo 2 puede variar a la baja o al alza dependiendo del tamaño de las lecturas y del ratio de mutaciones.

Por otra parte, el Cuadro 4.10 muestra el tiempo del procesamiento para las diferentes etapas de cada uno de los flujos de trabajo para el conjunto de datos de 100 nts con un ratio de mutaciones de 0.1 %, 1 % y 2 %. Si se compara el tiempo tiempo total de procesamiento con el obtenido por la suma de todas las etapas, el primero es ligeramente superior debido a las comunicaciones entre la etapas. Como se puede observar, cuando el ratio de mutaciones aumenta, las etapas C y D tienen una carga de trabajo más elevada en el primer flujo de trabajo. Además, en este caso, el número de lecturas que pasa al segundo flujo de trabajo es mayor, lo que incrementa el tiempo de procesamiento de la etapa B del último flujo. Cabe destacar que la etapa que conlleva un coste más elevado es la etapa C del primer flujo de trabajo.

A continuación se compara la velocidad y la sensibilidad de HPG Aligner ARN SA+M frente a tres alineadores muy utilizados: TopHat 2+Bowtie 2 (v2.0.10+2.1.0) [50], MapSplice (v2.1.5) [54], y STAR (v2.3.0e) [12]. En el Cuadro 4.11 se muestran los parámetros utilizados para la ejecución de estos alineadores. En todos los casos se han realizado pruebas para determinar el número óptimo de hilos, pero solo se muestran los resultados correspondientes a la mejor configuración, es decir, a la que ha conseguido el menor tiempo de ejecución. En general, esto corresponde al uso de t=12 hilos. Debido a su elevado tiempo de ejecución, TopHat 2+Bowtie 2 (v2.0.10+2.1.0) y MapSplice solo se han evaluado para lecturas de 100 nt.

El Cuadro 4.12 y la gráfica superior izquierda de la Figura 4.6 muestran el tiempo de ejecución y la sensibilidad de los cuatro alineadores con lecturas de 100 nts y distintos ratios de mutaciones. Como se puede ver, los tiempos de ejecución de MapSplice y de TopHat2 son muy

superiores a los de HPG Aligner SA+M y STAR. Por este motivo, para los tamaños de lecturas más grandes tan solo se presentan los resultados obtenidos por estos dos últimos alineadores (en el resto de gráficas de la Figura 4.6 y en el Cuadro 4.13). Como se puede ver, la diferencia temporal es favorable para HPG Aligner ARN SA+M, y crece con el ratio de mutaciones y, especialmente, con el tamaño de lectura.

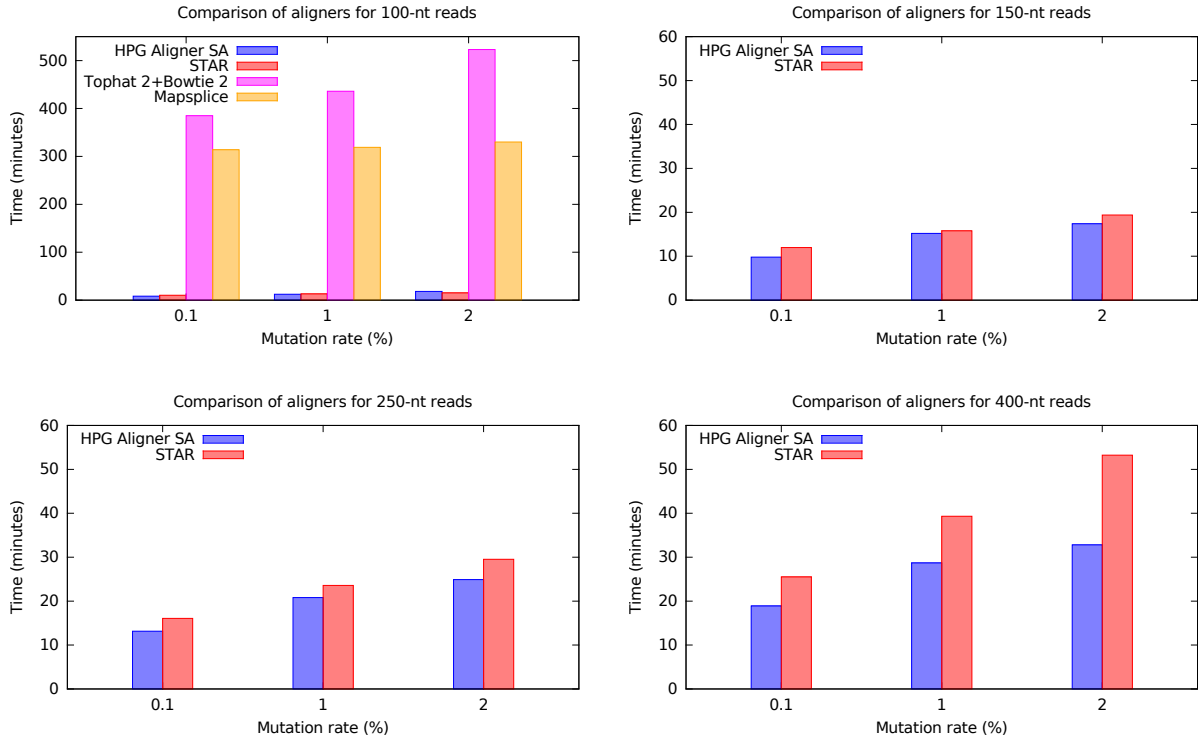


Figura 4.6: Rendimiento computacional (tiempo de ejecución en minutos) de los cuatro alineadores de ARN con distintos tamaños de lecturas y ratios de mutaciones

Por otro lado, los Cuadros 4.12 y 4.13 muestran la sensibilidad de los alineadores usando dos métricas: el porcentaje de lecturas mapeadas (columna RM) y el porcentaje de lecturas correctamente mapeadas (RCM). El Cuadro 4.12 muestra la sensibilidad obtenida por los cuatro alineadores con los tres bancos de pruebas de lecturas de 100 nts y distintos ratios de mutaciones. Como puede observarse, para los ratios de mutación de 1 % y 2 %, el par de alineadores TopHat 2+Bowtie 2 ofrecen porcentajes de mapeo inferiores a los proporcionados por HPG Aligner ARN SA+M, STAR y MapSplice. Estos tres obtienen unos resultados muy similares en la métrica RM. En cambio, si nos fijamos en la columna RCM, HPG Aligner ARN SA+M y MapSplice obtienen porcentajes superiores a STAR.

El Cuadro 4.13 muestra la sensibilidad para los dos alineadores más rápidos, HPG Aligner ARN SA+M y STAR, con los bancos de pruebas de lecturas de 150, 250 y 400 nts y distintos ratios de mutaciones. En este punto hay que enfatizar que, dado que los dos mapeadores ofrecen unos ratios similares de mapeo de lecturas (RM), la clave es cómo de correctos son los alineamientos (RCM). Los resultados muestran que HPG Aligner ARN SA+M consigue un mayor porcentaje de RCM que STAR.

El Cuadro 4.14 y 4.15 analiza la sensibilidad de los alineadores centrándose en la métrica RCM, usando las lecturas simuladas clasificadas en 5 subconjuntos. Los cuatro primeros correspon-

Tamaño	Ratio	HPG Aligner			STAR		
lecturas (nts)	mutaciones (%)	RM	RCM	T	RM	RCM	T
100	0,1	99,20	96,00	8,30	99,30	91,39	10,22
	1	98,80	94,80	12,30	99,20	86,70	13,07
	2	97,20	92,52	18,20	97,76	81,48	15,20
Tamaño	Ratio	MapSplice			TopHat 2		
lecturas (nts)	mutaciones (%)	RM	RCM	T	RM	RCM	T
100	0,1	99,60	95,87	314,00	97,90	96,20	385,00
	1	99,30	94,90	319,00	87,98	86,95	436,00
	2	97,70	91,30	330,00	63,49	62,77	523,00

Cuadro 4.12: Sensibilidad (RM y RCM en %) y tiempo de ejecución (en minutos) de los cuatro alineadores de ARN con lecturas de 100 nts y distintos ratios de mutaciones

Tamaño	Ratio	HPG Aligner			STAR		
lecturas (nts)	mutaciones (%)	RM	RCM	T	RM	RCM	T
150	0,1	99,30	96,20	9,80	99,64	91,96	11,98
	1	98,86	94,57	15,20	99,50	86,95	15,78
	2	98,21	93,76	17,40	99,29	82,50	19,38
250	0,1	98,90	93,95	13,15	99,40	90,39	16,08
	1	98,30	92,40	20,80	98,90	85,65	23,57
	2	97,70	91,34	24,90	98,70	80,38	29,53
400	0,1	98,60	92,00	18,90	99,26	89,23	23,50
	1	97,70	89,20	28,70	98,70	84,16	39,33
	2	94,70	86,20	32,80	95,40	64,49	53,23

Cuadro 4.13: Sensibilidad (RM y RCM en %) y tiempo de ejecución (en minutos) de HPG Aligner ARN SA+M y STAR con distintos tamaños de lecturas y ratios de mutaciones

Ratio mutaciones (%)	Exones	Lecturas (%)	HPG Aligner	STAR	MapSplice	TopHat 2
0,1	1	75,10	97,95	97,74	98,63	98,45
	2	22,62	92,19	76,04	88,66	90,14
	3	2,19	74,44	35,93	78,46	84,05
	>3	0,09	29,83	6,65	32,64	40,07
	m.e.	5,21	77,00	2,70	66,71	88,15
1	1	74,49	96,42	94,13	97,91	89,61
	2	23,09	91,67	69,02	87,28	79,49
	3	2,35	76,00	28,67	75,92	76,86
	>3	0,07	52,03	12,85	61,84	65,15
	m.e.	5,18	79,37	1,97	65,74	78,88
2	1	75,77	93,58	88,94	95,06	65,02
	2	22,09	90,91	61,80	81,43	56,06
	3	2,07	72,46	21,03	61,54	52,68
	>3	0,07	55,63	9,75	44,41	47,75
	m.e.	4,91	79,32	1,68	57,64	55,12

Cuadro 4.14: Sensibilidad (RCM en %) de los cuatro alineadores de ARN con lecturas de 100 nts y distintos ratios de mutaciones, en función del número de exones por lectura (1, 2, 3 o más)

Ratio		150 nts			250 nts			400 nts		
mutaciones (%)	Exones	Lecturas (%)	HPG Aligner	STAR	Lecturas (%)	HPG Aligner	STAR	Lecturas (%)	HPG Aligner	STAR
0,1	1	68,37	98,09	98,17	61,37	97,77	97,90	59,54	97,63	96,54
	2	23,99	94,92	85,74	15,82	93,35	89,02	8,63	91,89	87,76
	3	7,14	84,80	57,87	17,18	87,09	76,41	12,70	86,70	84,46
	>3	0,50	69,17	31,96	5,63	74,91	55,07	19,13	78,26	70,34
	m.e.	10,63	87,82	53,02	17,04	85,29	67,46	20,06	82,68	72,51
1	1	67,60	96,81	94,41	61,65	96,20	93,58	58,05	96,06	93,76
	2	24,42	92,53	79,16	15,53	91,46	84,24	8,86	89,16	82,17
	3	7,37	82,83	49,30	16,86	85,84	70,80	12,84	82,72	78,67
	>3	0,61	70,02	27,92	5,96	74,22	49,54	20,15	73,35	60,91
	m.e.	1,92	86,71	47,24	16,88	84,39	63,73	21,05	80,25	67,60
2	1	68,19	95,61	90,33	61,89	95,78	89,33	59,31	90,48	71,06
	2	24,26	92,71	73,80	15,51	88,99	77,70	8,90	85,85	63,49
	3	7,04	81,47	41,85	17,16	83,48	62,99	12,73	83,13	59,99
	>3	0,51	66,40	19,12	5,44	72,34	41,03	19,06	75,33	47,52
	m.e.	10,78	87,38	43,48	16,73	82,85	58,34	20,00	80,60	51,98

Cuadro 4.15: Sensibilidad (RCM en %) del alineador HPG Aligner ARN SA+M y STAR con distintos tamaños de lecturas y ratios de mutaciones, en función del número de exones por lectura (1, 2, 3 o más)

Tamaño lecturas (nts)	Ratio mutaciones (%)	#Puntos de unión	HPG Aligner	STAR	MapSplice	TopHat 2
100	0,1	58.367	95,64	95,13	92,80	94,67
	1	56.527	96,69	95,90	93,87	93,96
	2	57.113	96,04	95,30	92,32	89,89

Cuadro 4.16: Número de puntos de unión presentes en cada conjunto de datos y porcentaje de puntos de unión detectados por cada alineador

Tamaño lecturas (nts)	Ratio mutaciones (%)	#Puntos de unión	HPG Aligner	STAR
150	0,1	58.687	97,54	96,50
	1	59.040	97,65	96,43
	2	58.502	97,40	96,28
250	0,1	58.790	97,85	96,83
	1	59.776	98,00	96,77
	2	59.167	97,92	96,70
400	0,1	58.342	97,91	96,59
	1	59.305	98,32	97,30
	2	58.679	97,93	96,52

Cuadro 4.17: Número de puntos de unión presentes en cada conjunto de datos y porcentaje de puntos de unión detectados por HPG Aligner ARN SA+M y por STAR

den a las lecturas simuladas que contienen solamente 1, 2, 3 o más exones. Los últimos contienen las lecturas simuladas que tienen al menos un exón de 10 nts o menor (etiquetado como "m.e."). Estos cuadros muestran que HPG Aligner ARN SA+M consigue una mejor sensibilidad que STAR en todos estos casos y obtiene una sensibilidad mucho mejor que MapSplice y TopHat 2+Bowtie 2 en 8 de los 15 casos evaluados.

Finalmente, los Cuadros 4.16 y 4.17 muestran el número de puntos de unión presentes en cada conjunto de datos junto con el porcentaje de puntos de unión que han sido detectados por cada alineador. Estos resultados muestran que HPG Aligner ARN SA+M ha detectado más puntos de unión correctos que el resto de alineadores.

4.8 Evaluación del *framework*

En este apartado se presenta la sensibilidad y el rendimiento del *framework* con diferentes configuraciones, tanto para el alineamiento de secuencias de ADN, como de ARN.

Los experimentos se han realizado en el clúster Maverick del TACC (Texas Advanced Computing Center). Cada nodo de este clúster tiene dos 10-core Intel Xeon E5-2680 v2 Ivy Brdidge CPUs, 256 GiB de memoria RAM, y una tarjeta NVIDIA Tesla K40 Atlas GPU con 12 GiB de memoria G-RAM conectada vía PCI-e Gen 2. El clúster está interconectado mediante una red Mellanox FDR Infiniband. Utilizando esta misma red es posible acceder a un sistema de ficheros Lustre de 20 PiB.

La evaluación de los experimentos del *framework* para secuencias de ADN se ha realizado con un banco de pruebas *single-end* compuesto por 40 millones de lecturas de 100 nts, generado con el simulador dwgsim. El ratio de mutaciones se ha fijado en un 0.1 % con un 10 % de estas siendo inserciones y borrados. A este banco de pruebas se le ha denominado «D40M».

La evaluación de los experimentos del *framework* para secuencias de ARN se ha realizado con tres bancos de pruebas *single-end*: dos de ellos compuestos por 10 millones de lecturas y el tercero formado por 80 millones de lecturas, todos con secuencias de 100 nts. Los tres bancos de pruebas se han generado mediante el simulador BEERS. El ratio de mutaciones ha sido fijado a 0.1 % en uno de los ficheros de 10 millones de lecturas, «R40M0.1», y en el de 80 millones de lecturas, «R80M0.1»; para el segundo conjunto de datos de 10 millones, el ratio de mutaciones se ha incrementado al 2 %, «R10M2.0». La frecuencia de inserciones y borrados ha sido fijada al valor por defecto (0.05 %) para los tres bancos de pruebas.

	Mapeador	Líneas de comandos
DNA	Bowtie 2	`bowtie2 -x indexB2/hs -p 16 -S ./out/align.sam data.fq`
	Cushaw2-GPU	`cushaw2-gpu -r indexC2/hs.fa -f data.fq -t 16 -o ./out/align.sam`
	Cushaw3	`cushaw3 align -r indexC3/hs.fa -f data.fq -t 16 -o ./out/align.sam`
	HPG Aligner DNA SA	`hpg-aligner dna -i indexSA/ -f data.fq -cpu-threads 16 -o ./out/`
RNA	HPG Aligner RNA BWT+M	`hpg-aligner rna -i indexBWT/ -f data.fq -cpu-threads 16 -o ./out/`
	HPG Aligner RNA SA	`hpg-aligner rna -i indexSA/ -f data.fq -cpu-threads 16 -o ./out/`
	MapSplice 2	`mapsplice.py -c indexM/ -x indexM/hs -1 data.fq -p 16 -o ./out/`
	STAR	`STAR -genomeDir indexST/ -readFilesIn data.fq -runThreadN 16 -outFileNamePrefix ./out/`
	Tophat 2	`tophat2 -p 16 -no-convert-bam -no-sort-bam -o ./out/ iT/hs data.fq`

Cuadro 4.18: Comandos usados para la ejecución de los alineadores

Los parámetros establecidos para la ejecución de HPG Aligner BWT+M y SA+M durante la segunda etapa del *framework*, en caso de que alguno de estos dos alineadores sea el seleccionado en esta etapa, son los siguientes: el mínimo y máximo tamaño de intrón será de 40 y 500.000 nts, respectivamente; el mínimo tamaño de CAL ha sido fijado a 20 nts; y la configuración para SWA fue fijada a sus valores por defecto (acierto: 5, error: -4, abrir hueco: 10, extender hueco: 0,5).

Un paso previo al análisis de las prestaciones del *framework* ha sido el de evaluar por separado el rendimiento de los distintos alineadores multihilo, tanto de ADN como ARN, mostrados en el Cuadro 4.18. La configuración de lanzamiento de estos mapeadores se muestra en ese cuadro. Esta evaluación se ha realizado utilizando los 16 núcleos de un único nodo del clúster. Experimentalmente se ha comprobado que el número de hilos óptimo para la mayor parte de estos alineadores es justamente 16.

La parte superior del Cuadro 4.19 muestra el tiempo de ejecución para los distintos alineadores de ADN. La parte inferior del cuadro muestra el coste temporal para el procesamiento de secuencias de ARN, que varía entre 60 y 1.715 segundos. En los resultados puede observarse que existe una gran diferencia entre el tiempo de procesamiento de STAR/HPG Aligner y el del resto de alineadores. Cabe destacar que este estudio inicial no pretende comparar entre sí el rendimiento de los alineadores. De hecho, los resultados seguramente cambiarán si se utilizan otros bancos de pruebas diferentes, o si los parámetros del alineador se ajustasen para un conjunto de datos en particular. El principal objetivo del análisis es evaluar el *framework* y comprobar si puede utilizarse para reducir el tiempo de ejecución de un alineador conservando su sensibilidad.

La Figura 4.7 muestra la ganancia de velocidad obtenida con el *framework* configurado con una sola etapa, utilizando los alineadores de ADN mostrados en la parte superior del Cuadro 4.19, 12 nodos (16 hilos por nodo) y el banco de datos «D40M». Como puede observarse, la aceleración obtenida por la mayor parte de los alineadores es moderada quedándose en un factor aproximado de 6 sin alcanzar el máximo teórico de 12 para 12 nodos. El alineador HPG Aligner ADN SA, en cambio, tan solo alcanza un factor de aceleración de 3 para 12 nodos, obteniendo el peor resultado de entre todos los alineadores.

Por otro lado, la Figura 4.8 muestra las prestaciones que ofrece el *framework* para el alineamiento de ARN con los alineadores mostrados en la parte inferior del Cuadro 4.19 para los bancos de pruebas «R10M0.1» y «R80M0.1». En todas las ejecuciones los alineadores fueron ejecutados por el *framework* con la línea de comando mostrada en el Cuadro 4.18. Para este experimento no se activaron ni la lectura en paralelo de los datos de entrada, ni la mezcla de datos solapada. Como se puede ver en la figura, la ganancia de velocidad que se obtiene para MapSplice 2 y TopHat 2 es moderada, en cambio, es muy baja para STAR y las dos variantes de HPG Aligner ARN evalua-

Tipo	Mapeador	Tiempo
DNA/D40M	Bowtie 2	732
	Cushaw2-GPU	926
	Cushaw3	2.146
	HPG Aligner ADN SA+M	367
RNA/R10M0.1	HPG Aligner ARN BWT+M	65
	HPG Aligner ARN SA+M	66
	MapSplice 2	1.470
	STAR	60
	Tophat 2	1.715

Cuadro 4.19: Tiempo de ejecución (en segundos) de los alineadores multihilo, usando un solo nodo del clúster Maverick

das: BWT+M y SA+M. Por otra parte, la Figura 4.9 muestra el comportamiento del *framework* cuando se lanzan los alineadores STAR, HPG Aligner BWT+M y SA+M con el banco de pruebas «R80M0.1». Este experimento demuestra que aunque el tamaño de los experimentos aumente considerablemente, la escalabilidad apenas mejora.

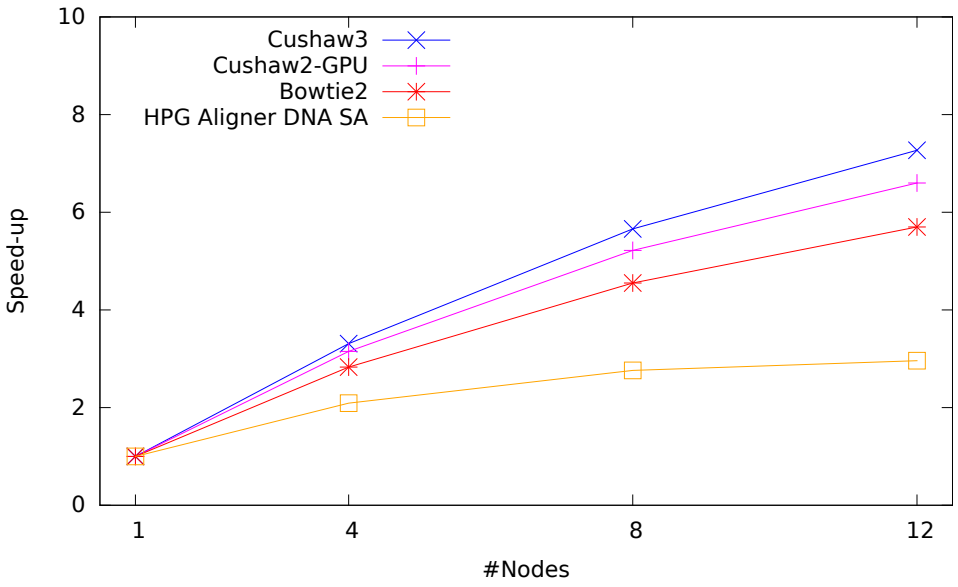


Figura 4.7: Aceleración de la primera etapa del *framework* usando alineadores de ADN y con el conjunto de datos «D40M» en función del número de nodos

Para averiguar qué parte del proceso estaba provocando un cuello de botella, se ha hecho un estudio detallado del tiempo de ejecución de las siguientes cuatro etapas de HPG Aligner BWT+M: la distribución de los datos de entrada, la carga del índice del genoma de referencia, el procesamiento de los resultados y la mezcla de los resultados. Los experimentos se lanzaron en uno y en 12 nodos. El Cuadro 4.20 muestra que el tiempo de mapeo se reduce desde 478,38 s para un nodo hasta 37,36 s para 12 nodos. Esto demuestra que para la etapa de mapeo, la escalabilidad es casi perfecta. En

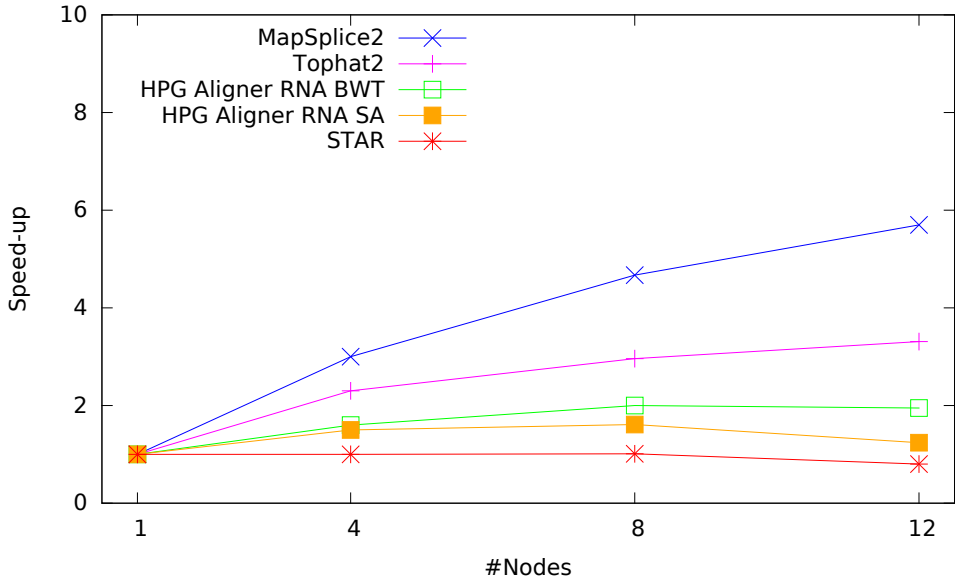


Figura 4.8: Aceleración de la primera etapa del *framework* usando alineadores de ARN y con el conjunto de datos «R40M0.1»

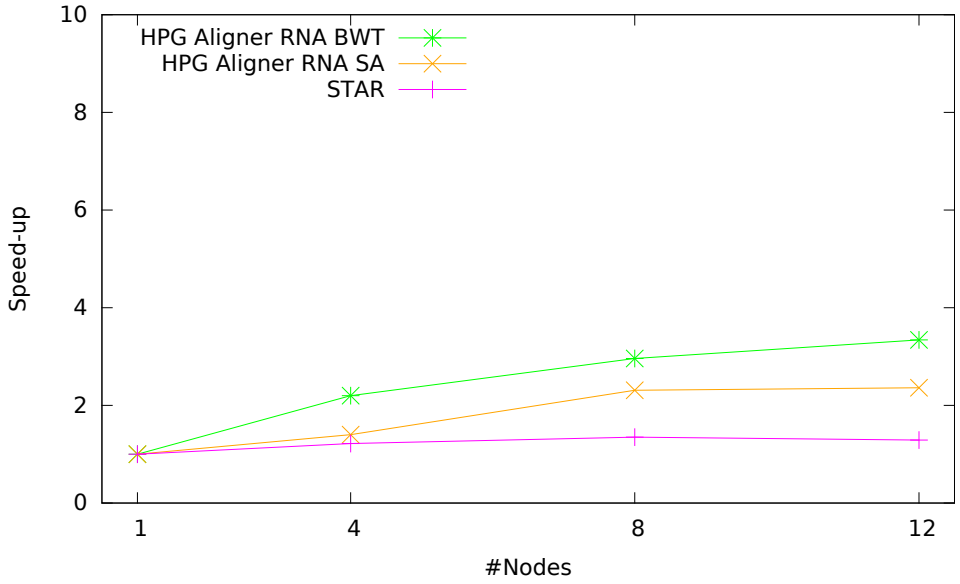


Figura 4.9: Aceleración de la primera etapa del *framework* usando los alineadores de ARN seleccionados para el estudio del conjunto de datos «R80M0.1»

cambio, este cuadro muestra que el 66 % del tiempo de ejecución total para 12 nodos se utiliza en la distribución de los datos de entrada y en la unión de los resultados generados. Estos resultados indican claramente que estas dos etapas deben de mejorarse poder acelerar el *framework*.

La ganancia obtenida por el *framework* cuando se activan las dos opciones encargadas de optimizar estas dos etapas se muestran en la Figura 4.10. En la gráfica puede observarse un in-

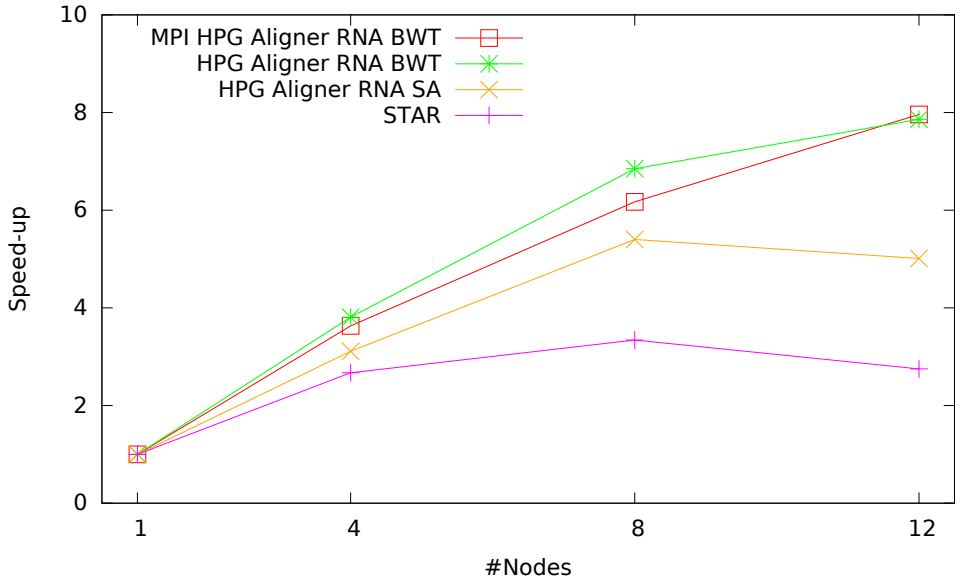


Figura 4.10: Aceleración de MPI de HPG Aligner RNA BWT y de la primera etapa del *framework* mejorado usando los alineadores de ARN seleccionados para el estudio del conjunto de datos «R80M0.1»

#Nodos	División entrada	Carga índice	Procesamiento	Unión resultados	Tiempo total
1	–	14,95 (3,0 %)	478,38 (97,0 %)	–	493,33
12	55,25 (37,4 %)	12,67 (8,6 %)	37,36 (25,3 %)	42,41 (28,7 %)	147,69

Cuadro 4.20: Tiempo de ejecución (en segundos) de la primera etapa del *framework* con HPG Aligner ARN BWT+M, con 1 y 12 nodos. Los números entre paréntesis indican el porcentaje del coste total

cremento significativo de la aceleración. Curiosamente, ese modo de funcionamiento del *framework* cuando se ha seleccionado el alineador HPG Aligner BWT+M muestra un patrón de escalabilidad similar al generado por HPG Aligner ARN BWT+M+MPI. Esto es un indicativo de que el *framework* está ofreciendo un rendimiento muy cercano a la versión programada con MPI: HPG Aligner ARN BWT+M+MPI.

El siguiente estudio analiza la sensibilidad de los alineadores cuando se ejecutan a través del *framework*. Un problema potencial cuando se realiza el procesamiento de los datos en un clúster es la pérdida de sensibilidad. Esta perdida es debida a la división del trabajo entre los nodos, lo que conlleva que la información de los alineamientos ya realizados también se divida entre estos. La fase de unión de la información que se realiza después de la primera etapa tiene por objeto solucionar esta problemática, ya que hace que todos los nodos compartan la misma información. Además, la posibilidad de combinar dos alineadores diferentes en el *framework*, también debería mejorar la sensibilidad de los resultados obtenidos.

El Cuadro 4.21 muestra la sensibilidad y el tiempo de ejecución del *framework* aplicado a un problema de ADN con uno y cuatro nodos. Como puede observarse, la sensibilidad no varía de forma significativa cuando intervienen varios nodos en la ejecución. Además, con cuatro nodos, el

	Un nodo			Cuatro nodos		
Alineador	Mapeadas	Correctas	Tiempo	Mapeadas	Correctas	Tiempo
Bowtie 2	99,12	94,65	732	99,12	94,65	258
Cushaw2-GPU	99,94	90,09	926	99,94	90,09	293
Cushaw3	100,00	92,69	2.146	100,00	92,69	647
HPG Aligner DNA SA	99,91	98,74	367	99,91	98,74	175

Cuadro 4.21: Sensibilidad (en %) y tiempo de ejecución (en segundos) del *framework* con varios alineadores de ADN con uno y cuatro nodos

porcentaje de lecturas correctamente alineadas se mantiene y el tiempo de ejecución se reduce en todos los alineadores. Una segunda etapa, en este caso, el del alineamiento de secuencias de ADN, es inútil, ya que todas las lecturas se almacenan como mapeadas en la primera etapa.

El Cuadro 4.22 muestra un estudio similar, para el caso de alineamiento de secuencias de ARN. Para este análisis, se seleccionaron los siguientes cuatro alineadores para la primera fase del *framework*: HPG Aligner ARN BWT+M, MapSplice 2, STAR, y TopHat 2, que se combinaron con: HPG Aligner ARN BWT+M, MapSplice 2, y STAR en la segunda fase. El banco de pruebas sobre el que se han lanzado los experimentos es «R10M2.0».

Los resultados obtenidos muestran los beneficios de activar la fase de refinamiento, ya que se observa un importante incremento de lecturas mapeadas correctamente. En ambos casos, con uno y cuatro nodos, cuando se utiliza STAR en la primera fase, el porcentaje de lecturas mapeadas durante la segunda fase decrece ligeramente con respecto a la primera. Esto es debido a que algunas de las lecturas que STAR considera como mapeadas, para los criterios de HPG Aligner ARN BWT+M y MapSplice 2 no lo son, ya que sus criterios son mucho más estrictos que los de STAR. En cualquier caso, en todos los casos, el porcentaje de lecturas correctamente mapeadas se incrementa en la fase de refinamiento.

Por último, el Cuadro 4.23 muestra el tiempo de ejecución con uno y cuatro nodos cuando la segunda fase se activa con el alineador HPG Aligner ARN BWT+M. En este caso, el incremento de tiempo es perfectamente aceptable en ambos escenarios, con uno y cuatro nodos. Además, la aceleración obtenida con 4 nodos es significativa con MapSplice 2 y TopHat 2, aunque bastante más reducida con STAR.

a) Solo un nodo multihilo

	Sin segunda fase		Con segunda fase					
			HPG Aligner		MapSplice 2		STAR	
Alineador	Mapeadas	Correctas	Mapeadas	Correctas	Mapeadas	Correctas	Mapeadas	Correctas
HPG Aligner	99,00	93,25	—	—	99,72	93.68	99,70	93,70
MapSplice 2	98,50	92,05	99,42	94,05	—	—	99,40	93,50
STAR	98,86	82,38	98,64	88,62	98,59	87.69	—	—
TopHat 2	63,86	63,26	99,07	95,35	98,86	91.53	99,01	89,18

b) 4 nodos multihilo

	Sin segunda fase		Con segunda fase					
			HPG Aligner		MapSplice 2		STAR	
Alineador	Mapeadas	Correctas	Mapeadas	Correctas	Mapeadas	Correctas	Mapeadas	Correctas
HPG Aligner	98,97	93,12	—	—	99,71	93,56	99,70	93,61
MapSplice 2	98,45	90,74	99,38	93,37	—	—	99,35	92,70
STAR	98,86	82,38	98,64	88,68	98,56	87,08	—	—
TopHat 2	63,69	63,01	99,06	95,26	98,79	89,18	99,01	89,00

Cuadro 4.22: Sensibilidad (en %) y tiempo de ejecución (en segundos) del *framework* con un alineador de ARN en la primera fase, con y sin segunda fase con otro alineador, con uno y cuatro nodos

	Un nodo multihilo		4 nodos multihilo	
Alineador	Una fase	Dos fases	Una fase	Dos fases
MapSplice 2	1.299	1.343	436	451
STAR	79	165	69	87
TopHat 2	2.163	2.275	856	892

Cuadro 4.23: Tiempo de ejecución (en segundos) del *framework* con distintos alineadores de ARN con uno y cuatro nodos multihilo, con y sin segunda fase con HPG Aligner ARN BWT+M

Parte II

Epistasis y FaST-LMM

CAPÍTULO 5

Introducción

Este capítulo introduce la problemática relacionada con la detección de la epistasis y enumera las diferentes aplicaciones existentes para su análisis. El trabajo se centra en el estudio de FaST-LMM, una de las aplicaciones más conocidas en este campo debido a su sensibilidad. Este estudio ha consistido en el análisis del rendimiento para detectar los principales cuellos de botella del módulo de FaST-LMM que se encarga del estudio de la epistasis.

5.1 Descripción del problema

Desde que en 1990 se inauguró el Proyecto Genoma Humano [19], la continua evolución de las tecnologías desarrolladas para estudiar el genoma ha permitido disponer de herramientas de análisis genético capaces de procesar una cantidad masiva de datos en un breve periodo de tiempo. Esta capacidad de cómputo ha propiciado la aparición de diversos estudios genéticos.

En genética, un estudio de asociación del genoma completo [24] (en inglés, GWAS —Genome-wide association study—, o WGAS —Whole genome association study—) es un análisis de una variación genética a lo largo de todo el genoma humano con el objetivo de identificar su asociación a un rasgo observable. Entre estos, el de determinadas enfermedades. Conocer qué variaciones genéticas se relacionan con determinadas enfermedades es sumamente importante, ya que serviría para investigar cómo mejorar su prevención, diagnóstico y tratamiento [1, 10].

El problema de los estudios GWAS es que la mayoría de las enfermedades son complejas y tienen patrones de herencia ambiguos, generalmente formados por la combinación de genes que se encuentran en diferentes zonas genómicas [59, 52, 17]. Pese a que inicialmente se consideraba que cada gen hacía su propia contribución característica en el marco de un conjunto de otros genes, en general, el efecto de cualquier gen depende de muchos otros genes.

Cuando la expresión de un determinado carácter fenotípico es debida a la interacción entre diferentes genes, esta interacción recibe el nombre de epistasis. Así pues, la epistasis tiene lugar cuando la acción de un gen se ve modificada por la acción de otros genes (p.e., cuando el efecto del gen que controla el color del pelo es ignorado al estar presente el gen que causa el albinismo).

Por otro lado, cuando en al menos un 1 % de la población se detecta que en una determinada posición del genoma hay unos pocos nucleótidos que cambian, esta variación recibe el nombre de polimorfismo de nucleótido simple (SNP, en inglés, de Simple Nucleotide Polymorphism). Los

SNP constituyen hasta el 90 % de todas las variaciones genómicas humanas, y aparecen cada 1.300 bases en promedio, a lo largo del genoma humano. Aunque los SNP por sí mismos no proporcionan información sobre genes específicos; sirven para indicar una localización cromosómica que es probable que esté estrechamente relacionada con un determinado fenotipo.

Así pues, para encontrar las posibles interacciones entre genes, el análisis epistásico se centra en estudiar la relación entre determinados conjuntos de SNP en uno o en multitud de individuos que presentan un determinado rasgo o enfermedad. Este estudio es muy complejo y costoso desde el punto de vista computacional, principalmente debido al gran número de SNP existentes. Por este motivo, es necesario desarrollar aplicaciones capaces de cuantificar la interacción entre los distintos SNP en el menor tiempo y con la mayor precisión posible.

5.2 Estado actual

En los últimos años se han desarrollado diversos métodos estadísticos para modelar e identificar interacciones epistásicas entre variantes genéticas. Esto ha dado lugar al desarrollo de una gran variedad de aplicaciones que permiten realizar este análisis. El Cuadro 5.1 muestra algunas de ellas. Estas aplicaciones se pueden clasificar por diferentes criterios: según el orden de las interacciones que analizan, donde las aplicaciones de n-vías analizan interacciones entre «n» SNP; según el método estadístico que utilizan para detectar estas interacciones; y según el tipo de computación de altas prestaciones (HPC) que explotan.

Así pues, se pueden encontrar aplicaciones como BOOST [52], GBOOST [57], EpistSearch [15] y EpiGPU [18] que analizan la interacción entre pares de SNP, es decir, son aplicaciones de 2vías y que aplican un método de regresión lineal para detectar esta relación. Estas aplicaciones se caracterizan principalmente porque su coste computacional es moderado. GBOOST o EpistSearch están implementadas utilizando diferentes tipos de HPC, consiguiendo de esta forma reducir el tiempo de ejecución en relación al de BOOST y manteniendo la sensibilidad de los resultados. En concreto, GBOOST explota el paralelismo a nivel de GPU mientras que EpistSearch obtiene el paralelismo utilizando múltiples GPUs/FPGAs. Por otro lado, EpiGPU explota también el paralelismo a nivel de GPU, pero a diferencia de las otras dos, que están desarrolladas con CUDA, esta última está desarrollado con OpenCL.

Otras de las aplicaciones que se muestran en el Cuadro 5.1, como por ejemplo TEAM [58], se basan en minería de datos, lo que implica el estudio de una enorme cantidad de información para obtener resultados. Por otro lado, SNPRuler [53] se basa en ir aprendiendo a medida que avanza el procesamiento y va obteniendo resultados, de esta manera, es capaz de utilizar la información que va generando para tomar las siguientes decisiones.

Por último, si el objetivo principal de una aplicación que realice el análisis de epistasis es la sensibilidad de sus resultados, la más destacable es FaST-LMM [27]. El método que aplica estudia todas las posibles interacciones entre pares de SNP para detectar la mejor predicción posible del fenotipo. El principal inconveniente que presenta este método es su elevado coste computacional. Por esta razón, la implementación original de esta aplicación ofrece una versión para HPC que explota el paralelismo a nivel de procesos en sistemas multiprocesador. También ofrece una versión para sistemas clústeres formados por múltiples nodos, pero con el inconveniente que solo se puede ejecutar en Azure.

Así pues, de entre todas las aplicaciones mostradas en el Cuadro 5.1, FaST-LMM es muy popular entre la comunidad biológica, siendo una de las que más sensibilidad proporciona en el estudio de la epistasis. En este trabajo se ha seleccionado esta aplicación para realizar un

análisis detallado de su rendimiento y plantear diferentes optimizaciones que reduzcan su tiempo de procesamiento.

Software	Método	Tipo de paralelización	URL
BOOST	Regression	—	http://bioinformatics.ust.hk/BOOST.html#BOOST
Encore	Gen. linear model	OpenMP	https://github.com/insilico/encore
EpiGPU	Regression	OpenCL	https://github.com/explodecomputer/epiGPU
EpistSearch	Regression (BOOST)	CUDA/FPGA	—
FastEpistasis	Brute force	SMP/MPI	https://www.cog-genomics.org/plink/1.9/epistasis
FaST-LMM	Linear-mixed model	Hadoop	https://github.com/MicrosoftGenomics/FaST-LMM
GBOOST	Regression (BOOST)	CUDA	http://bioinformatics.ust.hk/BOOST.html#GBOOST 2.0
SNPRuler	Machine learning	—	http://bioinformatics.ust.hk/SNPRuler.zip
SUPER	Linear-mixed model	—	—
TEAM	Data mining	—	http://www.csbio.unc.edu/epistasis/download.php

Cuadro 5.1: Algunas de las aplicaciones para el análisis de epistasis de 2vías

5.3 Análisis de FaST-LMM para el estudio de epistasis

FaST-LMM es un aplicación que tiene como propósito general realizar el procesamiento GWAS y, entre otros estudios, puede llevar a cabo el análisis de epistasis. Así pues, el módulo integrado en FaST-LMM encargado de este análisis obtiene la relación epistásica entre cualquier combinación de pares de SNP. Este procesamiento consiste en calcular para cada posible pareja, el p-valor del chi-cuadrado asociado con la diferencia entre la probabilidad del logaritmo de una hipótesis nula y otra alternativa. Este método, como ya se ha indicado, proporciona una sensibilidad muy alta pero con un coste computacional muy elevado.

La aplicación está desarrollada en Python y utiliza bibliotecas específicas y eficientes para el cálculo numérico y manejo de datos (por ejemplo, numpy y pandas) que permiten un procesamiento de altas prestaciones.

Además, FaST-LMM explota el paralelismo en sistemas multiprocesador creando múltiples procesos para el procesamiento de los datos. Esto presenta varios inconvenientes en la implementación: incrementa la dificultad en el manejo de la memoria y de las variables compartidas y requiere que los procesos intercambien información.

A continuación se describe con detalle el método utilizado en el módulo encargado del estudio de la epistasis integrado en FaST-LMM 0.2.31. Además, se realiza un análisis destallado del rendimiento de esta parte la aplicación, mostrando sus principales cuellos de botella [32].

5.3.1 Algoritmo

Desde el punto de vista algorítmico, el método que se aplica para cuantificar la interacción entre los pares de SNP se divide en tres etapas que denotaremos S1, S2 y S3, tal y como puede observarse en la Figura 5.1. A continuación se describe cada una de ellas:

Etapa S1: Conlleva la ejecución consecutiva de las siguientes operaciones.

S1.1. Genera la matriz G de efectos aleatorios de $n \times k$, donde n es el número de individuos y k el número de SNP que analizar. Si el número de SNP es mayor que el número de individuos ($k \geq n$), se procesa la covarianza de la matriz de efectos aleatorios $K = GG^T$.

S1.2. Según el número de individuos y SNP a analizar se realizan distintas operaciones:

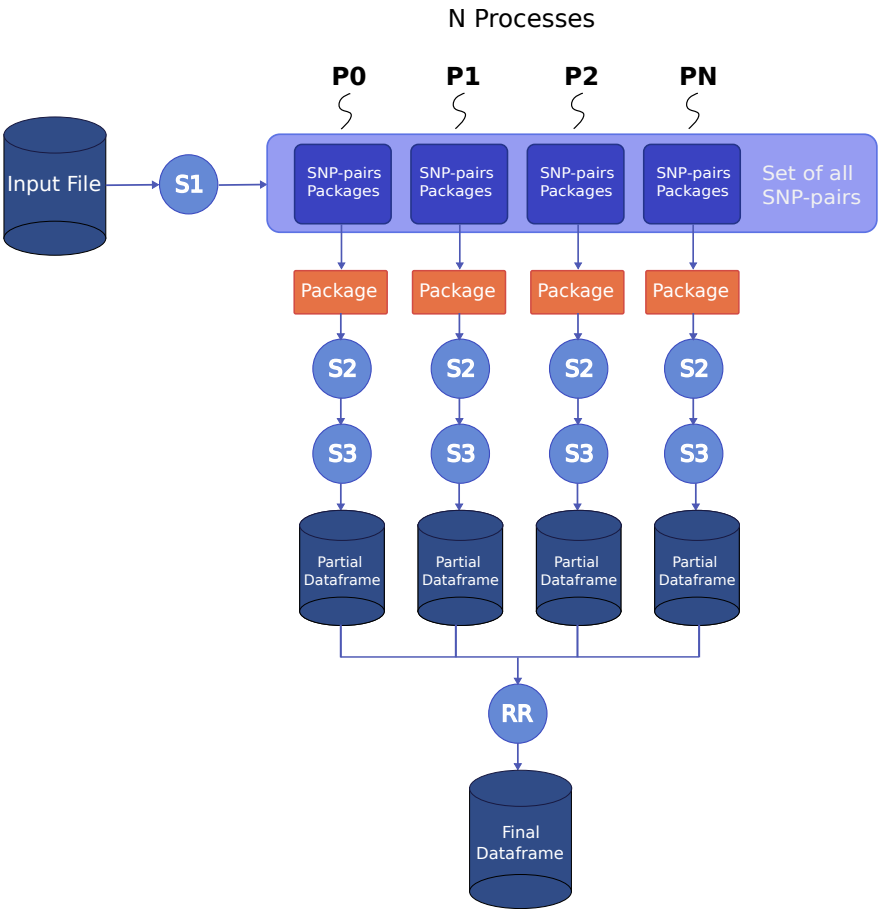


Figura 5.1: Flujo de trabajo del módulo de epistasis de FaST-LMM

Si el número de SNP es menor que el número de individuos ($k < n$), se procesa la descomposición: $G = U\Sigma V^T$, donde Σ es una matriz diagonal $k \times k$ que contiene los valores singulares de G ordenados de forma descendente; a continuación, se fija $S = \Sigma^2$.

Si el número de SNP es mayor o igual al número de individuos ($k >= n$) se realiza la descomposición espectral: $K = USU^T$ donde S es una matriz diagonal de $n \times n$ que contiene los valores propios de K. En ambos casos, las columnas de las matrices U, V de $n \times r$ con $r = \text{mín}(n,k)$ son ortonormales.

Después de completar esta parte del procesamiento, el conjunto de las posibles parejas de SNP se divide en paquetes de tamaño fijo (por defecto, 1.000 pares). A continuación, se crean tantos procesos como hayan sido fijados por el usuario en el momento del lanzamiento de la ejecución de la aplicación y los paquetes se reparten entre estos procesos. Cada proceso recibirá los paquetes asignados y ejecutará, en paralelo con los restantes procesos, las etapas S2 y S3, calculando el p-valor asociado a cada pareja de SNP.

Etapa S2: Realiza los siguientes cálculos matriciales para cada uno de los paquetes asignados:

S2.1. Calcula $M_{UX} = U^T X$, donde las filas de la matriz X de $r \times m$ están asociadas con los individuos; y las columnas corresponden a: i) las covarianzas; ii) el número de los alelos menores para cada SNP diferente en el paquete que se está procesando; y iii) el producto de los alelos menores para cada pareja de SNP de dicho paquete. Como el número de SNP diferentes varía en cada paquete, la dimensión m será diferente de un paquete a otro; aunque siempre será al menos tres veces el número de parejas de SNP del paquete más el número de covarianzas consideradas (el valor del cual fue fijado a 1 en nuestros experimentos).

SS2.2. Si el número de SNP es menor que el número de individuos (i.e., $k < n$), se procesa $M_{UUX} = X - UM_{UX}$, de otra forma, M_{UUX} está vacío.

Etapa S3: Esta etapa realiza las siguientes operaciones para cada pareja de SNP del paquete.

S3.1. Calcula la probabilidad del logaritmo de la hipótesis seleccionando las columnas de X, $U^T X$, y M_{UUX} que corresponden a la covarianza y a cada uno de los dos SNP.

S3.2. Calcula la probabilidad del logaritmo de la hipótesis alternativa seleccionando las columnas X, $U^T X$, y M_{UUX} que corresponden a la covarianza, a cada uno de los dos SNP, y al producto de esos dos SNP.

S3.3. Calcula el p-valor del test chi-cuadrado para la diferencia de los valores previos de la probabilidad del logaritmo.

S3.4. Almacena los resultados como una fila nueva dentro de la estructura *dataframe*.

5.3.2 Análisis del rendimiento

En este apartado se analiza el rendimiento del módulo integrado en FaST-LMM para el estudio de epistasis. Para la realización de los experimentos se han formado diferentes conjuntos de datos que se han extraído del WTCC (Welcome Trust Case Control Consortium) para la enfermedad del trastorno bipolar. Este estudio contiene un total de 455.086 SNP (k) para 4.804 individuos (n). Debido al elevado coste computacional que supone la aplicación de este método, las pruebas se han realizado sobre una fracción pequeña del conjunto original de SNP, consistentes en los siguientes

		SNP					
p	t	2000	3000	4000	5000	6000	7000
16	1	343.46	1111.33	2558.80	2418.03	3477.87	4741.93
8	2	422.47	1192.14	2842.32	2658.02	3578.57	5569.05
4	4	446.40	1339.54	3328.72	2896.50	4500.83	6947.21
2	8	652.62	1980.10	4361.08	4161.01	5994.55	7862.61
1	16	862.64	2454.95	5194.58	5035.87	7300.73	9970.29

Cuadro 5.2: Tiempo de ejecución (en segundos) de la etapa **S2** usando los conjuntos de datos para 2.000, 3.000, 4.000, 5.000, 6.000 y 7.000 SNP, y diferentes números de procesadores (p) e hilos (t)

tamaños k=2.000, 3.000, 4.000, 5.000, 6.000 y 7.000 SNP. Cabe destacar que, como FaST-LMM divide el trabajo en paquetes que comprenden 1.000 pares de SNP cada uno y los tamaños de las matrices están limitados por el número de individuos, $k >= n$, cuando el número de SNP crece, el tiempo de ejecución se incrementa de forma lineal con respecto al número de pares totales del paquete ($>= k^2/2$). Por lo tanto, las conclusiones extraídas a partir de los experimentos presentados, pueden extrapolarse fácilmente al conjunto de datos original.

Los experimentos han sido realizados en un servidor compuesto por 2 Intel Xeon E5-2620v4 con 8-procesadores, 32 GiB de memoria RAM, y una GPU NVIDIA P100 Pascal. El sistema operativo es RedHat Linux 2.6.32. El software utilizado para la realización de los experimentos ha sido: FaST-LMM 0.231, Python 2.7.13, PyCUDA 2016.1.2, Scikit-CUDA 0.5.1, Intel MKL Update 11 (icc 17.0.1), y NVIDA CUBLAS 7.5.

A continuación se describe el análisis que se ha llevado a cabo. En primer lugar se han analizado dos aspectos muy importantes para las prestaciones de la aplicación.

Análisis del grado de paralelismo: Consiste en el estudio de la mejor configuración entre procesos/hilos a partir de los procesadores disponibles en el sistema multiprocesador que permite obtener un rendimiento óptimo. Por un lado, el usuario, en el momento de iniciar la ejecución de FaST-LMM puede especificar el grado de paralelismo a nivel de procesos indicando el número de estos que quiere que se creen. Cuanto mayor es el número de procesos que intervienen en la ejecución, menor será el número de paquetes de SNP que se tienen que procesar en las etapas S2 y S3. Por otro lado, como se ha comentado anteriormente, FaST-LMM utiliza la biblioteca *numpy* para realizar de forma eficiente los cálculos matriciales que se necesitan en las subetapas S2.1, S2.2 del algoritmo y en la obtención de los valores propios en S1.2. Esta biblioteca contiene algunos *kernels* matemáticos como BLAS (basic linear algebra subprograms) que forman parte de la biblioteca MKL de Intel y utiliza múltiples hilos para llevar a cabo estos cálculos matriciales. Así pues, este estudio ha consistido en obtener la distribución óptima de los procesadores del sistema entre los procesos que se tienen que crear y los hilos que se utilizarán para realizar dichas operaciones matriciales. El resultado de estos experimentos se muestra en el Cuadro 5.2 donde pueden observarse las diferentes combinaciones analizadas de procesos e hilos con diversos conjuntos de datos de distintos tamaños. Como conclusión, puede observarse que la configuración óptima son 16 procesos y 1 hilo. Teniendo en cuenta estos resultados, se han fijado estos parámetros de ejecución para el resto de experimentos.

Estudio del tamaño de paquete: Consiste en el análisis del tamaño de paquete óptimo para mejorar las prestaciones. Las parejas de SNP que forman parte del procesamiento se dividen en paquetes con un número fijo de pares. Es importante remarcar que el rendimiento de los *kernels* computacionales que permiten realizar las operaciones matriciales incluidas en la etapa S2 depende fuertemente del tamaño las matrices implicadas. A su vez, las dimensiones de estas matrices están ligadas al tamaño del paquete de SNP y al número de individuos incluidos en el estudio. En particular, si las dimensiones son muy pequeñas, el coste de mover los datos a través de la jerarquía de memoria y el sobrecoste de la ejecución paralela afectan negativamente al rendimiento. Así pues, se han realizado diversos experimentos usando diferentes tamaños de paquetes para determinar las prestaciones obtenidas por FaST-LMM para cada uno ellos. Como conclusión se puede decir que, con una configuración de 16 procesos y 1 hilo, el tamaño de paquete que fija por defecto FaST-LMM proporciona un ratio cercano al 92 % del pico teórico de rendimiento. Por esta razón, se ha concluido que la partición fijada por defecto en FaST-LMM proporciona unos tamaños de matrices lo suficientemente grandes como para obtener un rendimiento óptimo en las operaciones anteriormente mencionadas.

El siguiente paso ha consistido en analizar el coste temporal de cada una de las etapas del algoritmo y comprobar dónde se encuentra el principal cuello de botella. El estudio experimental anterior nos ha llevado a utilizar 16 procesos con un solo hilo por proceso, además de fijar el tamaño de paquete por defecto en 1.000 pares. Como ya se ha comentado, el conjunto de datos utilizado para realizar el experimento se compone de una serie de fragmentos de datos extraídos del conjunto original. Estos subconjuntos están compuestos por 2.000, 3.000, 4.000, 5.000, 6.000 y 7.000 SNP de 4.804 individuos.

La Figura 5.2 muestra la distribución del tiempo de ejecución a través de las diferentes etapas y las operaciones que se realizan en ellas. Esta gráfica muestra que la fase en la que se almacenan los resultados en la estructura *dataframe* (S3.4) es el principal cuello de botella. Por este motivo los esfuerzos del estudio de optimización se han centrado en esta parte. Además, la figura muestra la distribución del tiempo de ejecución a través de las etapas S1-S3. La etapa adicional denominada RR es la encargada de unir los resultados de todos los procesos creados en una única estructura *dataframe*. Estos resultados muestran que el coste de las etapas S1 y RR es despreciable frente al resto. En concreto, el coste de RR está por debajo del 0.6 % del total para cualquier numero de SNP, y el coste de la etapa S1 disminuye respecto al resto de etapas a medida que aumenta el número de SNP. Esto implica que el tiempo de ejecución de las etapas S2 y S3 siempre consumen más del 99 % del tiempo total. Por este motivo, se han excluido las etapas S1 y RR de los siguientes análisis.

Además, en la Figura 5.2 también se detallan los costes de las etapas S2 y S3. Para la etapa S2 se puede observar como el coste de ejecución corresponde principalmente a las multiplicaciones matriz-matriz que se llevan a cabo en las etapas S2.1 y S2.2. Para la etapa S3, las operaciones críticas corresponden a la etapa S3.1 (hipótesis nula), S3.2 (hipótesis alternativa), y especialmente a la S3.4 (almacenamiento de resultados). Las operaciones restantes en esta etapa constituyen un factor menor con respecto al coste total.

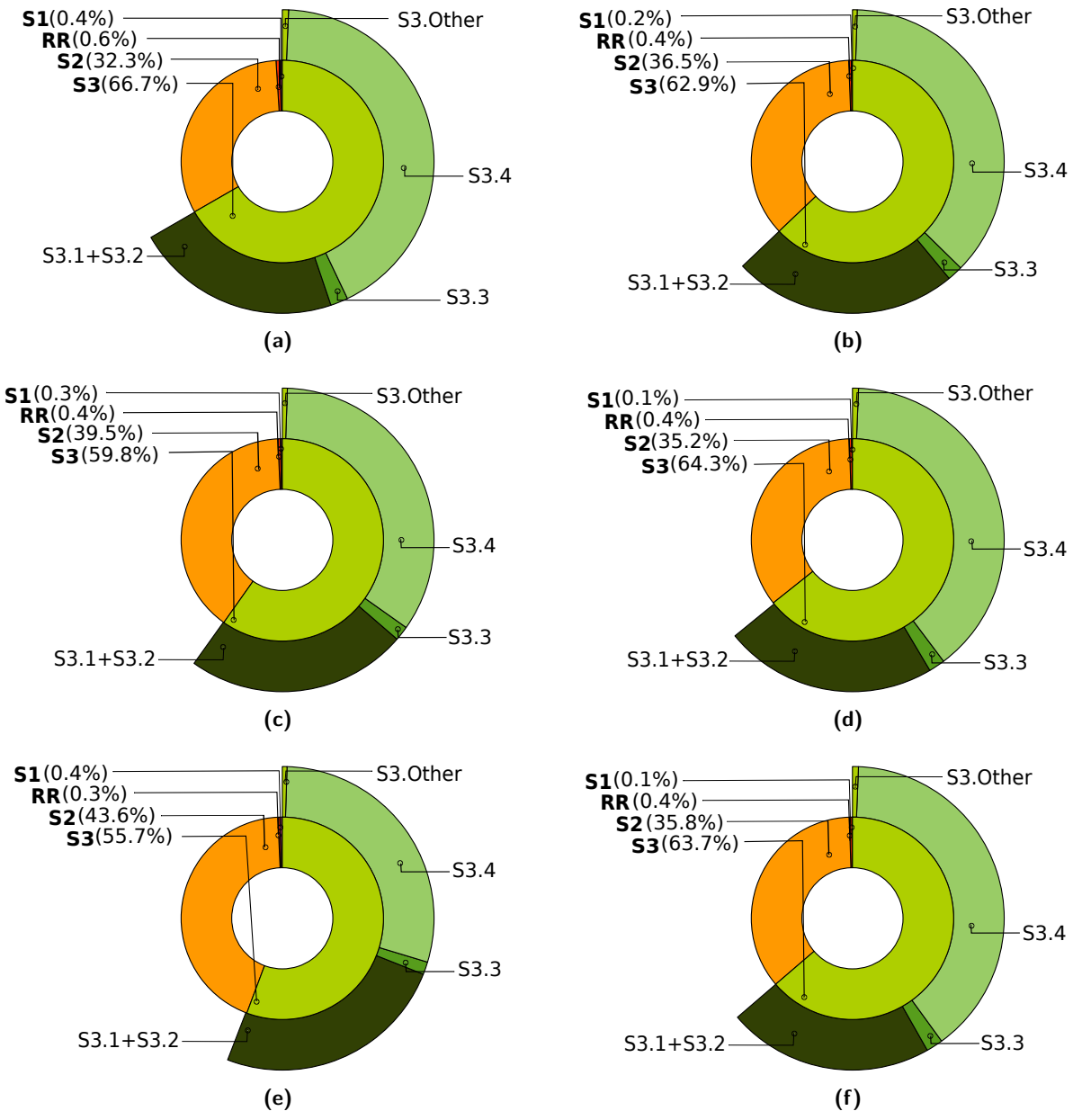


Figura 5.2: Distribución del tiempo a través de las etapas de FaST-LMM (usando 16 procesos). Las gráficas de la parte izquierda corresponden a los casos con: a) 2.000 SNP, 3.000 SNP, y e) 4.000 SNP; la de la parte derecha, a los casos con: b) 5.000 SNP, d) 6.000 SNP, y f) 7.000 SNP.

CAPÍTULO 6

Mejoras desarrolladas

En este capítulo se presentan las mejoras e implementaciones que se han realizado a partir del código original del módulo de FAST-LMM para el análisis de la epistasis. También se describen los experimentos realizados para analizar el impacto de las mismas frente al código inicial.

6.1 Introducción

El objetivo principal de estas mejoras es la reducción del tiempo de procesamiento, puesto que se trata de un software con un coste computacional muy elevado. Estas nuevas mejoras se basan, por un lado, en modificar el código original optimizando aquellas partes que suponen un cuello de botella y que se identificaron en el capítulo anterior, y por otro, se ha desarrollado una versión paralela para un clúster de computadores que, además de tener en cuenta las mejoras anteriores, permite ejecutar este código de forma distribuida reduciendo notablemente su tiempo de ejecución.

El estudio realizado mostró que las partes del algoritmo donde se encontraban los principales cuellos de botella estaban fundamentalmente en las etapas S2 y S3. Nuestra versión propone modificaciones en estas dos etapas manteniendo la sensibilidad del software original. En esta versión también se ha incorporado la posibilidad de utilizar un coprocesador gráfico. La idea aquí es que el coprocesador se encargue de realizar algunas de las operaciones de la etapa S2, que requieren mucha potencia de cálculo. La implementación propuesta consigue una aceleración cercana a 7,5 en un sistema multiprocesador equipado con una GPU.

Estas mejoras se han completado con una implementación paralela que se ejecuta sobre un clúster de computadores de forma distribuida. Las mejoras realizadas siguen manteniendo la sensibilidad del software original a la vez que ofrecen una aceleración cercana a la lineal sobre las plataformas en las que se ha probado su rendimiento. Esta implementación se ha basado en el estándar MPI para programación paralela y garantiza una distribución equilibrada de la carga de trabajo entre todos los procesos. En último lugar, en esta versión paralela se ha añadido una extensión al código que permite la utilización de múltiples GPU por nodo para poder descargar aún más a los procesadores del trabajo correspondiente al cálculo de operaciones matriciales de la etapa S2.

6.2 Optimizaciones del algoritmo original para un nodo multiprocesador

Este apartado se centra en las optimizaciones que se han aplicado sobre el código original de análisis de epistasis de FaST-LMM y que ha sido diseñado para sistemas multiprocesador. El aumento de rendimiento se obtiene de las mejoras realizadas en las etapas S2 y S3. Concretamente, estas modificaciones han consistido en almacenar los resultados intermedios en un diccionario Python en lugar de directamente sobre la estructura *dataframe*; en reorganizar el código de FaST-LMM; en reutilizar los datos para reducir el número de operaciones realizadas; y, finalmente, en procesar sobre la GPU aquellas partes del algoritmo que pueden calcularse más eficientemente en la GPU que en el procesador. A continuación se realiza una descripción detallada de cada una de estas optimizaciones.

6.2.1 *Dataframe*

En el estudio de las prestaciones del módulo original de epistasis se mostró que una de las partes más costosas del algoritmo ocurría en la etapa S3.4, y era debida al almacenamiento del resultado obtenido para cada pareja de SNP en una estructura de almacenamiento. En la versión original, la estructura que se utiliza es un *pandas Dataframe*. El manejo de esta estructura es muy similar a la de una base de datos, para cada operación de inserción de un nuevo elemento, se tiene que calcular su índice y después añadirlo, por lo que cada inserción conlleva un coste asociado.

La mejora ha consistido en utilizar una estructura intermedia que almacenará los resultados a medida que se obtienen. Esta estructura es un diccionario de Python que tiene un coste de inserción mucho menor. Así pues, en la nueva versión, los resultados intermedios se almacenan en este diccionario y una vez finalizado el procesamiento de todas las parejas de SNP, se copia el diccionario sobre la estructura *pandas DataFrame*. De esta forma, esta última estructura se inicializa y actualiza una sola vez.

6.2.2 Última columna y logaritmo de la probabilidad

Esta optimización se ha centrado en la etapa S3, fundamentalmente en el cálculo del logaritmo de las probabilidades de las hipótesis nula y alternativa para cada pareja de SNP. En ambos casos, como paso previo, se tienen que generar dos matrices, que más tarde intervienen en el proceso, una para la hipótesis alternativa y otra para la nula. Estas dos matrices únicamente difieren en la última columna. Concretamente, la matriz de la hipótesis alternativa incluye una columna extra con el producto de la pareja de SNP.

En el estudio previo del código original se ha detectado que estas matrices se generaban dos veces, una para cada hipótesis. La mejora ha consistido en reorganizar las operaciones de tal forma que en lugar de generar dos matrices se genera solo una de ellas. En concreto, en primer lugar se genera la la matriz correspondiente al cálculo del logaritmo de la probabilidad de la hipótesis alternativa y, a continuación, se reutiliza esta misma matriz, sin la última columna, para calcular el logaritmo de la probabilidad de la hipótesis nula. A esta optimización se le ha denominado «*Last Column*» («LC»).

En esta parte del procesamiento se ha implementado otra mejora basada en la reutilización de algunos resultados calculados previamente y cuyo coste de computo es elevado. Concretamente, al analizar el código original, se observó que como parte del cálculo del logaritmo de la probabilidad es necesario sumar y calcular los logaritmos de los elementos de un vector determinado. Puesto que el contenido de este vector no varía a lo largo del procesamiento, se ha modificado el código para

que este cálculo se realice una sola vez y luego se reutilice este resultado cada vez que se calcula el logaritmo de una probabilidad, a esta mejora se ha denominado «Logaritmo de Probabilidad» («LOG»).

6.2.3 Multiplicación de matrices en la GPU

Una de las partes del algoritmo que requieren más intensidad de cálculo esta localizada en la etapa S2 donde se realizan dos operaciones de multiplicación de matrices. Esta extensión carga sobre el procesador gráfico la realización de estas multiplicaciones para hacerlas de forma más eficiente.

Como se describió en el capítulo anterior, a partir de la etapa S2, el algoritmo permite realizar el procesamiento de distintos paquetes de pares de SNP en paralelo. Esto es, se crean un conjunto de procesos y cada uno, de forma independiente y en paralelo, procesa el resto de etapas (S2 y S3). Esto significa que en nuestra nueva propuesta, en la etapa S2, cada uno de estos procesos requerirá la GPU y descargará en ella la realización de dos multiplicaciones que se realizan en las subetapas S2.1 y S2.1.

Esta implementación puede generar problemas de procesamiento en la GPU si no se gestiona adecuadamente su memoria. Esto se produce porque pueden haber varios procesos que requieran la GPU simultáneamente y, por tanto, transfieran sus matrices a la memoria de la GPU. Entonces, puede ocurrir que no haya suficiente espacio para almacenar la información de todas las solicitudes que llegan a la vez. Para solucionar esta problemática, se ha desarrollado un gestor de memoria que controla las peticiones. Este gestor se encarga de comprobar, a medida que llegan las peticiones, si hay espacio suficiente en la memoria de la GPU para almacenar toda la información requerida.

Así pues, como parte del procesamiento de la etapa S2, cuando un proceso necesita lanzar una multiplicación matriz-matriz sobre la GPU, inicialmente calcula la cantidad de memoria que requiere dicha operación. A continuación, el proceso accede al gestor de memoria de la GPU para consultar cuánta memoria disponible hay en el dispositivo asociado. En el caso de haber suficiente memoria, el gestor registra la memoria consumida por esta nueva operación y notifica al proceso que puede acceder a la GPU para realizar esta operación. En caso contrario, es decir, cuando la memoria disponible no sea suficiente, el gestor de memoria bloquea al proceso hasta que haya liberado la memoria necesaria. Además, para evitar posibles desbordamientos de memoria en la GPU, el gestor se ha implementado como una región de exclusión mutua, donde únicamente puede haber un solo proceso accediendo a la vez.

Cuando el gestor de memoria autoriza la operación, las matrices de entrada son transferidas a la GPU y se realiza la multiplicación. Una vez finalizada, la matriz resultante se transfiere de vuelta al proceso. Finalmente, el proceso informa al gestor de memoria que la GPU ha finalizado la operación y este actualiza su cuenta de la memoria disponible en la GPU.

El objetivo de estas optimizaciones es que sean totalmente transparentes al usuario. Con esta finalidad, esta última extensión se ha implementado de tal forma que el usuario no tenga que recompilar el código fuente de FaST-LMM. Así pues, esta extensión se ha implementado en Python usando la biblioteca PyCUDA y el paquete scikit-cuda. PyCUDA ofrece una interfaz de NVIDIA CUDA para permitir el desarrollo de aplicaciones paralelas desde Python. Además, scikit-cuda provee una interfaz Python de alto nivel para el uso de muchas de las bibliotecas CUDA para la solución de problemas como son CUBLAS, CUFFT y CUSOLVER, entre otras.

En esta nueva implementación hay que tener en cuenta el tamaño del paquete de parejas de SNP, puesto que cuánto mayores sean las matrices que se procesan en la GPU, mejores serán las prestaciones obtenidas. En los experimentos del capítulo anterior se comprobó que la dimensión de las matrices de la etapa S2 dependía del tamaño del paquete. También se comprobó que en el

caso de un sistema multiprocesador sin GPU, el tamaño fijado por FaST-LMM era el óptimo. Es decir, aunque se aumentase el número de parejas de un paquete, no se mejoraban las prestaciones. Por el contrario, en esta extensión se ha comprobado que este tamaño influye notablemente, cuanto más grande es el tamaño de paquete, más grandes son las matrices, y es posible obtener mejores prestaciones al multiplicarlas en la GPU que con su tamaño inicial. En el apartado de resultados experimentales se detallan los experimentos realizados para fijar el tamaño de paquete óptimo.

6.3 Extensión multiGPU

En el apartado anterior se ha descrito la implementación basada en la utilización de una GPU y la problemática que esto conlleva con respecto a la gestión de la memoria y con las peticiones simultáneas que se puedan producir por parte de distintos procesos. La solución adoptada consistía en bloquear las peticiones hasta que pudieran ser atendidas. El objetivo de esta propuesta es mejorar este cuello de botella y reducir el tiempo de espera para acceder a la GPU. Para ello, se ha desarrollado una extensión que se puede ejecutar en sistemas multiprocesador equipados con varias GPU. Esta implementación permite realizar las multiplicaciones de distintos procesos en las distintas GPU del equipo.

La extensión multiGPU distribuye la carga de trabajo cíclicamente entre las GPU disponibles en el equipo. Esta distribución es estática y se realiza inicialmente aplicando la técnica de *Round-Robin*. Esto es, a cada uno de los procesos se le asigna una GPU de forma cíclica. Por ejemplo, su hubiese 4 procesos, los procesos 0 y 2 se vincularán a la GPU 0, mientras que los procesos 1 y 3, a la GPU 1. Después de esta inicialización, cada proceso crea su propio contexto con la GPU correspondiente.

La gestión en cada una de las GPU es similar a como se ha explicado en el apartado anterior. Se lanza un gestor de memoria para cada una de las GPU que controla las peticiones que llegan desde los distintos procesos que tienen asignada esa GPU. De forma que cada proceso, cuando precisa la GPU, solicita permiso a este controlador para que le autorice la realización de la operación, tal y como se ha explicado anteriormente.

6.4 Implementación distribuida

Tomando como base la implementación mejorada del módulo de epistasis de FaST-LMM que se ha descrito en los apartados anteriores, se ha desarrollado una versión distribuida capaz de explotar los recursos de un clúster de computadores. El principal objetivo de esta nueva implementación es reducir el tiempo de procesamiento aprovechando los recursos disponibles en la mayoría de los centros de cálculo actuales. Esta implementación, además de permitir que todos los nodos puedan trabajar en paralelo, también ha de asegurar que el reparto de la carga de trabajo sea lo más equilibrada posible. La comunicación entre los nodos se ha implementado utilizando MPI, que como ya se ha explicado previamente, es una interfaz de comunicación entre los nodos de un sistema basada en el paso de mensajes.

Esta propuesta crea inicialmente «np(+1)» procesos MPI (uno por cada nodo del clúster). Cada uno de estos procesos ejecutará el método descrito anteriormente sobre un conjunto distinto de paquetes de SNP. Así pues, cada proceso MPI inicia el procesamiento ejecutando la etapa S1 del algoritmo. Tras esta etapa, cada proceso MPI genera otros «p» procesos (en el nodo) que ejecutan simultáneamente las etapas S2 y S3 sobre los distintos paquetes de SNP asignados a este nodo. Los resultados obtenidos por cada proceso MPI se almacenan localmente. Es decir, cada proceso MPI recoge los resultados de todos los paquetes que se han procesado en ese nodo. A continuación, todos

los nodos envían estos resultados al nodo 0 para que genere una única estructura de almacenamiento con las relaciones entre todas las parejas de SNP.

En cualquier implementación paralela es fundamental la distribución de la carga de trabajo entre los procesos para que esta sea equilibrada. En este caso concreto, los paquetes de parejas de SNP se distribuyen, primero, entre los distintos procesos MPI, y después, entre los procesos internos que se crean en cada nodo. La forma de realizar este reparto ha dado lugar a dos versiones, una que aplica una distribución estática de la carga y otra que hace un reparto dinámico.

Estática. Consiste en repartir uniformemente el total de paquetes entre todos los procesos que se generan en el sistema. Así pues, el número de paquetes para cada proceso depende del total de paquetes y del número de procesos del sistema. En esta implementación paralela, el número total de procesos serán «np x p», donde «np» son los proceso MPI generados y «p» son los procesos que se crean en cada nodo. De esta manera, para un conjunto de datos con «k» SNP y un total de «w» procesos trabajadores, cada trabajador procesa sobre «k/(2wn)» paquetes, siendo «n» el tamaño del paquete en términos de parejas de SNP. Según esta solución, todos los nodos procesan la misma cantidad de paquetes y cada uno de ellos identifica el paquete que tiene que procesar en función del nodo en el que está, del total de procesos que hay en cada nodo, del total de parejas a procesar y del tamaño del paquete. Esta reparto es similar al que se hace en el método original pero, en este caso, teniendo en cuenta el nodo donde está el proceso que va a ejecutar el método.

Dinámica. Los paquetes de parejas de SNP son asignados a los procesos MPI bajo demanda. En esta solución se crean «np + 1» procesos MPI. El nodo 0 contiene dos procesos, un proceso trabajador y otro planificador que se encarga de controlar la asignación de paquetes entre los distintos trabajadores, como se muestra en la Figura 6.1. A su vez, cada proceso trabajador también tiene un gestor que se encarga de controlar si tiene paquetes para procesar. El proceso planificador inicialmente calcula el número total de paquetes a procesar e identifica el índice de fila y columna de los SNP que constituyen la primera pareja de cada paquete. A continuación, este planificador realiza una primera distribución de paquetes entre todos los trabajadores. El resto de paquetes no asignados esperarán en una cola en el planificador. El número de paquetes distribuidos inicialmente a cada trabajador es parametrizable. El gestor de paquetes de cada trabajador recibe los índices que delimitan la primera pareja de los paquetes a procesar en este proceso MPI, los almacena en una cola local y los asigna entre los distintos procesos que ha creado dentro de su nodo a medida que éstos finalizan el procesamiento de un paquete. Cuando el gestor de cada nodo detecta que no le quedan paquetes por procesar, solicita nuevos paquetes al planificador. Cuando la cola del planificador se vacía, el procesamiento finaliza. Debido al reducido sobrecoste de esta implementación, este método es preferible al anterior.

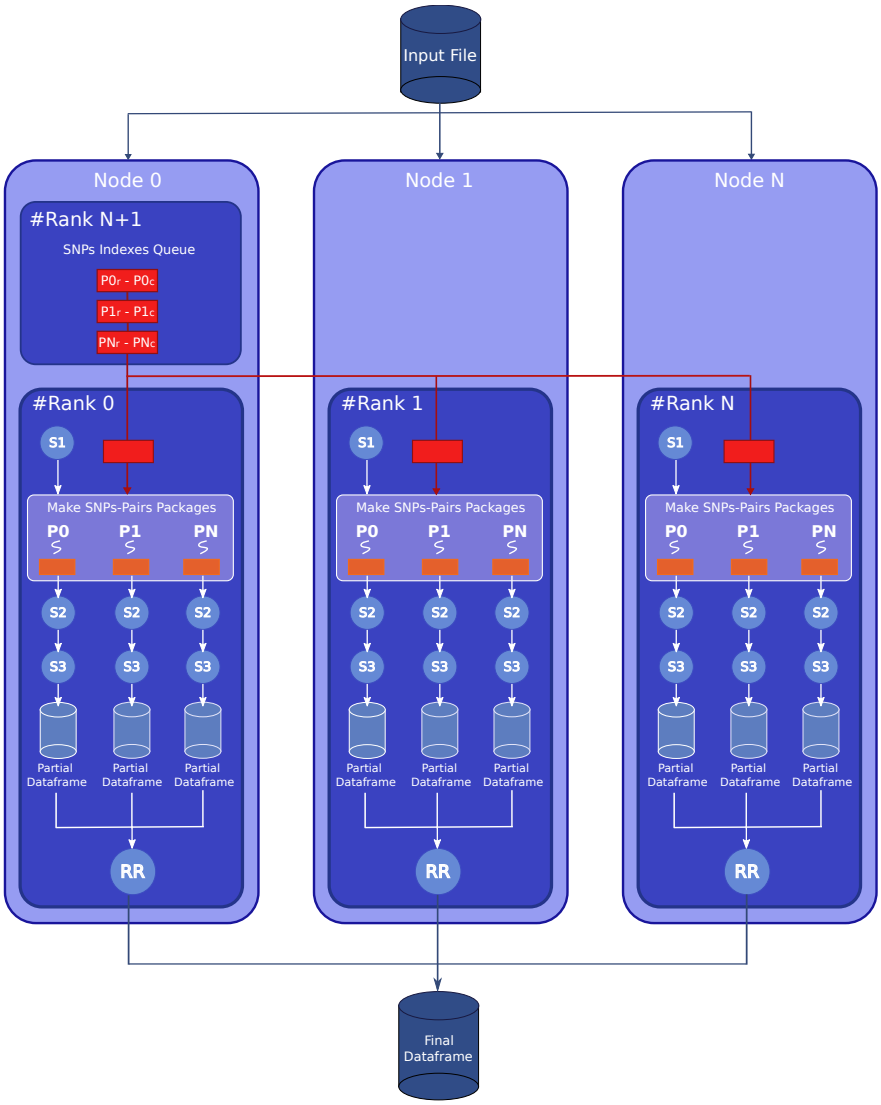


Figura 6.1: Flujo de trabajo de la extensión MPI del módulo de epistasis de FaST-LMM (versión con distribución dinámica)

6.5 Resultados experimentales

En este apartado se realiza el estudio experimental de las diferentes implementaciones que se han realizado: las mejoras para un único nodo multiprocesador equipado con una o varias GPU y para un sistema clúster.

Para los experimentos se ha utilizado una muestra de datos del WTCC (Welcome Trust Case Control Consortium) de la enfermedad desorden bipolar, que provee un total de 455.086 SNP para n = 4.804 individuos. Para reducir los tiempos de ejecución, se han usado una serie de subconjuntos de datos del conjunto original. Estos subconjuntos están formados desde k = 2.000 hasta 32.000 SNP (o lo que es lo mismo desde 1.999.000 hasta 511.984.000 parejas de SNP). Estos subconjuntos de datos reducen las muestras de SNP pero son lo suficientemente grandes como para evaluar adecuadamente las distintas propuestas. Esto es debido a que FaST-LMM divide el trabajo en paquetes que contienen 1.000 SNP cada uno (tamaño fijado por defecto) y, como ya se

ha comentado anteriormente, las pruebas con conjuntos de datos mucho más elevados conlleva un incremento lineal del tiempo de ejecución en relación al número de parejas de SNP. Por lo tanto, las conclusiones de los experimentos realizados son directamente extrapolables a conjuntos de datos más grandes.

Los sistemas utilizados para lanzar los experimentos han sido:

- Un nodo del servidor Tintorrum de la Universidad Jaume I de Castellón, equipado con 2 procesadores Intel Xeon E5-2620v4 8-core, 32 GiB de memoria RAM, y una GPU NVIDIA P100 «Pascal». El sistema operativo es una RedHat Linux 2.6.32.

- El servidor Piz Daint del Centro Supercomputacional de Suiza (CSCS) que contiene 5.272 nodos, cada uno con un Intel Xeon E5-2690 v3 a 2,60 GHz con 12 procesadores, y 64 GiB de memoria RAM y una NVIDIA Tesla P100 con 16 GiB. El sistema está interconectado mediante la red propietaria Aries de Cray, utilizando la topología de red «Dragonfly».

- Por último, el servidor Minotauro del Centro de Supercomputación de Barcelona (BSC). Es un servidor heterogéneo con 39 nodos, cada uno equipado con dos GPUs NVIDIA K80, 2 Intel Xeon E5-2630 v3 (Haswell) con 8 procesadores (cada uno a 2,4GHz y con 20 MiB L3 cache), y 128 GiB de memoria principal. El servidor está interconectado empleando una red Infiniband Mellanox FDR.

En cuanto al software utilizado, este ha sido: FaST-LMM 0.2.31, Python 2.7.13, PyCUDA 2016.1.2, Scikit-CUDA 0.5.1, Intel MKL Update 11 (ICC 17.0.1), y NVIDIA CUBLAS 8.0. Las versiones MPI han sido: MPI Slurm 17.02.7 en Piz Daint y Open MPI 1.6.5 en Minotauro.

Los experimentos se han realizado cada uno 5 veces y se ha calculado el coeficiente de variación (CV) [41], que además de estar siempre por debajo del 10 %, decrece rápidamente conforme aumenta el numero de SNP. Por ejemplo, con 6.000 o más SNP, el CV está como máximo al 3 %, y en la mayoría de los casos, al 1 %.

En primer lugar se ha evaluado el rendimiento de las mejoras realizadas en el módulo original de FaST-LMM identificadas como «D4D», «LC», «LOG» y «GPU» en un nodo de Tintorrum, utilizando un total de 16 procesos (uno por procesador). En la Figura 6.2 pueden observarse las prestaciones obtenidas con estas optimizaciones para los diferentes conjuntos de datos. La figura refleja una aceleración para la optimización «D4D» que ronda el 1,85 con respecto a la versión original de FaST-LMM 0.2.31 cuando k >= 4.804. Por otro lado, la optimización conjunta de «D4D+LC+LOG» no logra mejorar sustancialmente los resultados obtenidos por «D4D».

Para analizar el rendimiento de la extensión con GPU se ha modificado el tamaño de paquete con respecto al utilizado en los experimentos anteriores. Como ya se comentó, cuanto mayor es el tamaño de paquete, más grandes son las matrices que se multiplican y más prestaciones se obtienen en la GPU. Como puede observarse en el Cuadro 6.1, se han realizado distintos experimentos con un fichero de 5.000 SNP para diferentes tamaños de paquete. Estos experimentos se han lanzado en un nodo de Tintorrum de la Universidad Jaume I de Castellón con las optimizaciones «D4D+LC+LOG+GPU» de FaST-LMM. En el cuadro puede observarse que el resultado para 8.000 parejas de SNP no ha podido obtenerse. Esto ha sido debido a que no había suficiente memoria RAM en el nodo para la cantidad de datos requerida. En base a estos resultados, se ha fijado el tamaño óptimo de paquete en Tintorrum en 6.000 pares de SNP.

En la Figura 6.2 pueden observarse los resultados obtenidos conforme se van acumulando optimizaciones hasta llegar a la optimización etiquetada con «GPU». Estos resultados muestran que con $k \geq n$, se obtiene un factor de aceleración de entre 7,30 y 7,90 respecto a la implementación original. Como se ha remarcado anteriormente, el tiempo de procesamiento de FaST-LMM conforme

Tamaño de paquete	Tiempo (en s)
1.000	1.548,36
2.000	1.209,10
4.000	1.006,73
6.000	924,63
8.000	—

Cuadro 6.1: Tiempo de ejecución de FaST-LMM con las optimizaciones «D4D+LC+LOG+GPU» con un conjunto de datos de 5.000 SNP y diferentes tamaños de paquetes. No hay resultados para el tamaño de paquete de 8.000 SNP debido a que el servidor se quedó sin memoria

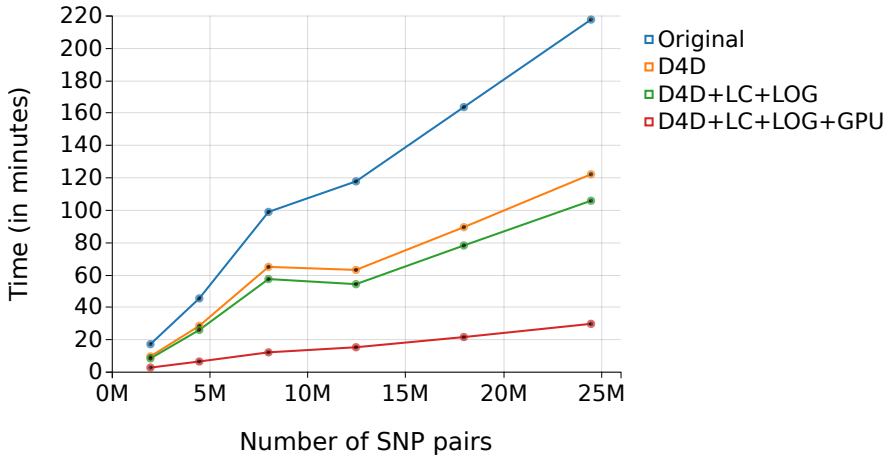


Figura 6.2: Tiempo de ejecución de FaST-LMM y sus optimizaciones con diferentes números de parejas de SNP (usando 16 procesos).

se incrementa el número de SNP crece linealmente con respecto al número de parejas. Esto permite concluir que el factor de aceleración se mantiene en el mismo orden cuando el número de SNP crece por encima de los 7.000.

Podemos concluir el análisis de las mejoras incluidas en el módulo de FaST-LMM para sistemas multiprocesador indicando que el rendimiento de la aplicación muestra que las subetapas S3.1 y S3.2 representan más del 54 % del tiempo total del procesamiento, generando el cuello de botella actual.

A continuación se muestra el análisis de rendimiento de la implementación paralela. Dado que este análisis se va a realizar en Minotauro y en Piz Daint, en primer lugar se ha evaluado la aceleración de cada una de las mejoras propuestas en uno de los nodos de cada uno de estos servidores. Las pruebas de la versión multiGPU se han realizado únicamente en Minotauro, ya que es el único de los dos que dispone de 2 GPU por nodo.

La Figura 6.3 muestra el tiempo de ejecución obtenido con los diferentes conjuntos de datos y la correspondiente aceleración para las diferentes mejoras implementadas sobre FaST-LMM, en un único nodo de Minotauro y de Piz Daint. Como se ha descrito anteriormente, y puede observarse en las Figuras 6.3 a) y b), cuando el número de SNP que se procesan es mayor que el número de individuos, el tiempo de ejecución crece linealmente con respecto al número de parejas de SNP. Cuando se procesan 8.000 SNP, las mejoras añadidas a la versión original de FaST-LMM reducen

el tiempo de ejecución de 9 horas y 13 minutos a 1 hora y 12 minutos en Piz Daint, y de 4 horas y 50 minutos a 41 minutos en Minotauro.

Las Figuras 6.3 c) y d) muestran que aplicando las mejoras descritas anteriormente, la aceleración ronda un 7,6 en Piz Daint y un 5,5 en Minotauro. La aceleración de FaST-LMM en Piz Daint es mayor que la de Minotauro debido a que el mayor ahorro de tiempo viene dado por la optimización de la GPU, y como Piz Daint dispone de una GPU más potente que la de Minotauro, en el primero se obtiene una aceleración mayor que en el segundo.

La Figura 6.3 d) muestra que aplicando la nueva versión multiGPU en Minotauro, la aceleración se incrementa de 5,5 a 6,9 de media. Aunque puede parecer una aceleración pequeña (de 1,25), hay que remarcar que la multiplicación matriz-matriz de la etapa S2 es la única beneficiada de esta extensión.

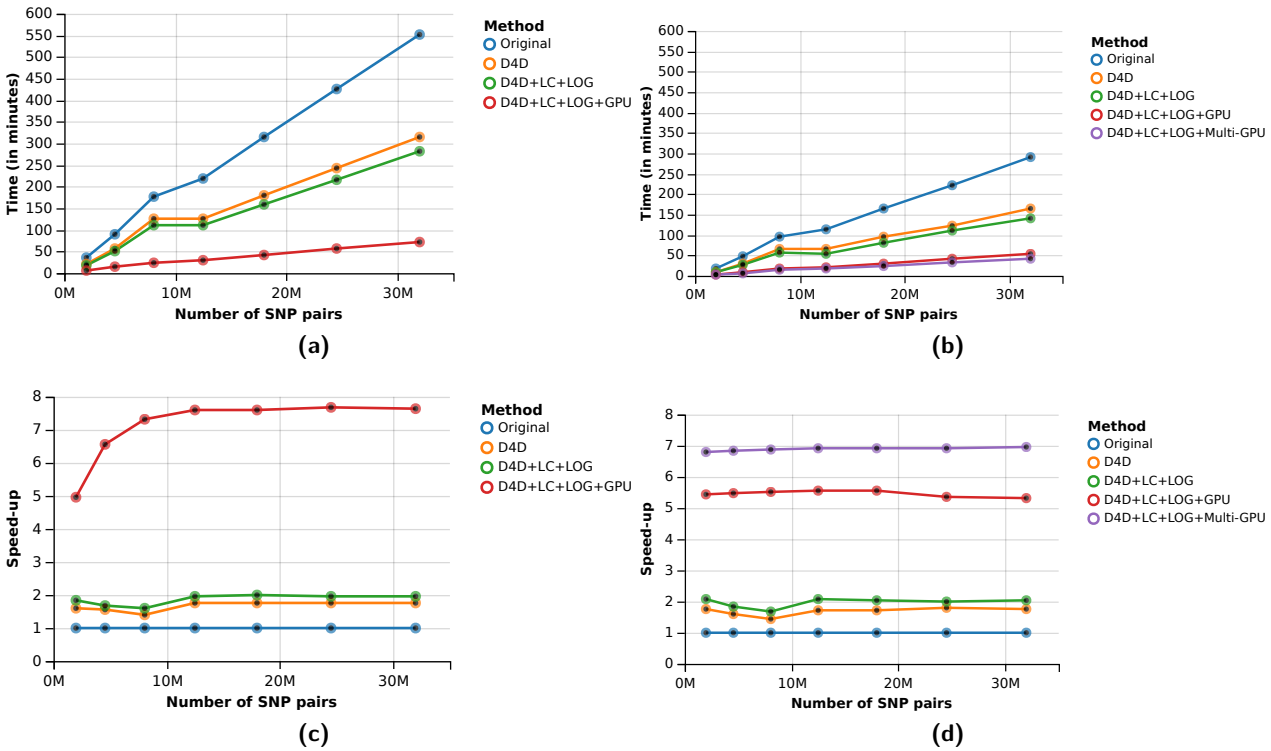


Figura 6.3: Tiempo de ejecución de la versión original de FaST-LMM para epistasis y de FaST-LMM con las mejoras anteriormente comentadas más la extensión multiGPU, y sus correspondientes aceleraciones, con diferentes números de parejas de SNP en un único nodo: con 12 procesos en Piz Daint (figuras «a» y «c»); y con 16 procesos en Minotauro (figuras «b» y «d»)

A continuación se muestran los resultados obtenidos en las dos implementaciones paralelas de FaST-LMM para clústeres utilizando las distribuciones de la carga, estática y dinámica. Para llevar a cabo los experimentos se han usado diferentes conjuntos de datos de diversos tamaños. Además, ambas implementaciones se han lanzado con diferentes números de nodos, tanto en Piz Daint como en Minotauro, para poder estudiar la escalabilidad de la aplicación.

El Cuadro 6.2 muestra los resultados de comparar la versión de distribución estática frente a la dinámica para un conjunto de datos de 16.000 SNP en el clúster Piz Daint. En el Cuadro 6.2 se muestran los tiempos de ejecución mínimo y máximo de los nodos que han intervenido en el procesamiento. Los resultados muestran que la versión estática genera una pequeña desviación del

balance de la carga de trabajo. Esta variabilidad viene dada por el uso compartido de la(s) GPU entre todos los procesos de un nodo. En comparación, el reparto dinámico produce una distribución del trabajo más equilibrada y el sobrecoste debido al reparto es despreciable comparado con el tiempo de ejecución de la aplicación. En este punto podemos concluir que cuando los nodos del clúster sean similares, los tiempos de las dos versiones serán parecidos. Sin embargo, en servidores con nodos heterogéneos, la versión dinámica obtendrá resultados mucho mejores.

	Distribución estática			Distribución dinámica		
Nodos	Mínimo	Máximo	Diferencia	Mínimo	Máximo	Diferencia
2	8.482,62	8.555,93	73,31	8.525,11	8.528,32	3,21
4	4.271,68	4.359,87	88,19	4.299,37	4.300,76	1,39
8	2.140,83	2.240,32	99,49	2.219,90	2.223,12	3,22
12	1.465,22	1.558,30	93,08	1.521,50	1.524,34	2,84
16	1.109,62	1.192,63	83,01	1.171,14	1.174,06	2,92

Cuadro 6.2: Tiempos máximo y mínimo por nodo (en segundos), y su diferencia, comparando la distribución de trabajo dinámica con la estática para la implementación clúster de FaST-LMM. El conjunto de datos usado ha sido de 16.000 SNP y se ha ejecutado en la plataforma Piz Daint

Con el objetivo de estudiar la escalabilidad fuerte, se ha ejecutado la versión dinámica para clúster de FaST-LMM utilizando entre 1 a 16 nodos en Piz Daint y en Minotauro con dos conjuntos de datos compuestos por 8.000 y 16.000 SNP. El tiempo de ejecución de estos experimentos y su correspondiente aceleración se recoge en la Figura 6.4. Conviene matizar que la aceleración mostrada es relativa a la versión mejorada de FaST-LMM presentada anteriormente. Cabe destacar que esta versión mejorada explota todos los recursos de cada nodo. Al ejecutar la extensión para clústeres de FaST-LMM con 16.000 SNP en Piz Daint, el tiempo se reduce de 4 horas 39 minutos con 1 nodo, a 20 minutos cuando se ejecuta con 16 nodos. Esta mejora temporal corresponde a una aceleración de 14,1. Por otro lado, al ejecutar la extensión para clústeres con 16.000 SNP con 16 nodos en Minotauro, se obtiene una aceleración similar, de 14,6. En ambos casos, como se puede ver en la Figura 6.4, la aceleración obtenida con 8.000 SNP es cercana a la ideal cuando se utilizan 8 o menos nodos. También se puede observar que al aumentar el tamaño del problema, al pasar de 8.000 a 16.000 SNP, la aceleración se mantiene cercana a la ideal hasta con 16 nodos. A la vista de estos resultados, es de esperar que con un mayor número de SNP, la aceleración se mantenga cercana a la ideal con un número mayor de nodos.

Por último, para evaluar desde otro punto de vista la escalabilidad de la extensión para clústeres, también se ha estudiado la escalabilidad débil. En este estudio, se ha evaluado cómo varía el tiempo de ejecución a medida que se incrementa el número de nodos utilizados mientras la carga de trabajo por nodo se mantiene constante. Para este propósito, como el tiempo de ejecución es lineal con respecto al número de parejas de SNP, se ha fijado el tamaño del problema por nodo a 31.996.000 parejas de SNP (3.000 SNP), y se ha variado el número de nodos entre 1 y 16. Los resultados obtenidos se muestran en el Cuadro 6.3. La media del tiempo de ejecución por nodo en Piz Daint ha sido de 4.281 segundos con una desviación estándar de 21 segundos. El mismo experimento en Minotauro tarda 259.073 segundos con una desviación estándar de 152 segundos. Por tanto, en ambos casos la aplicación está demostrando que permite resolver un problema mayor en el mismo tiempo sin más que añadir nuevos nodos. Extrapolando estos resultados, si se quisiera resolver un problema de un tamaño mucho mayor con k = 500.000 SNP, este tardaría 50 meses

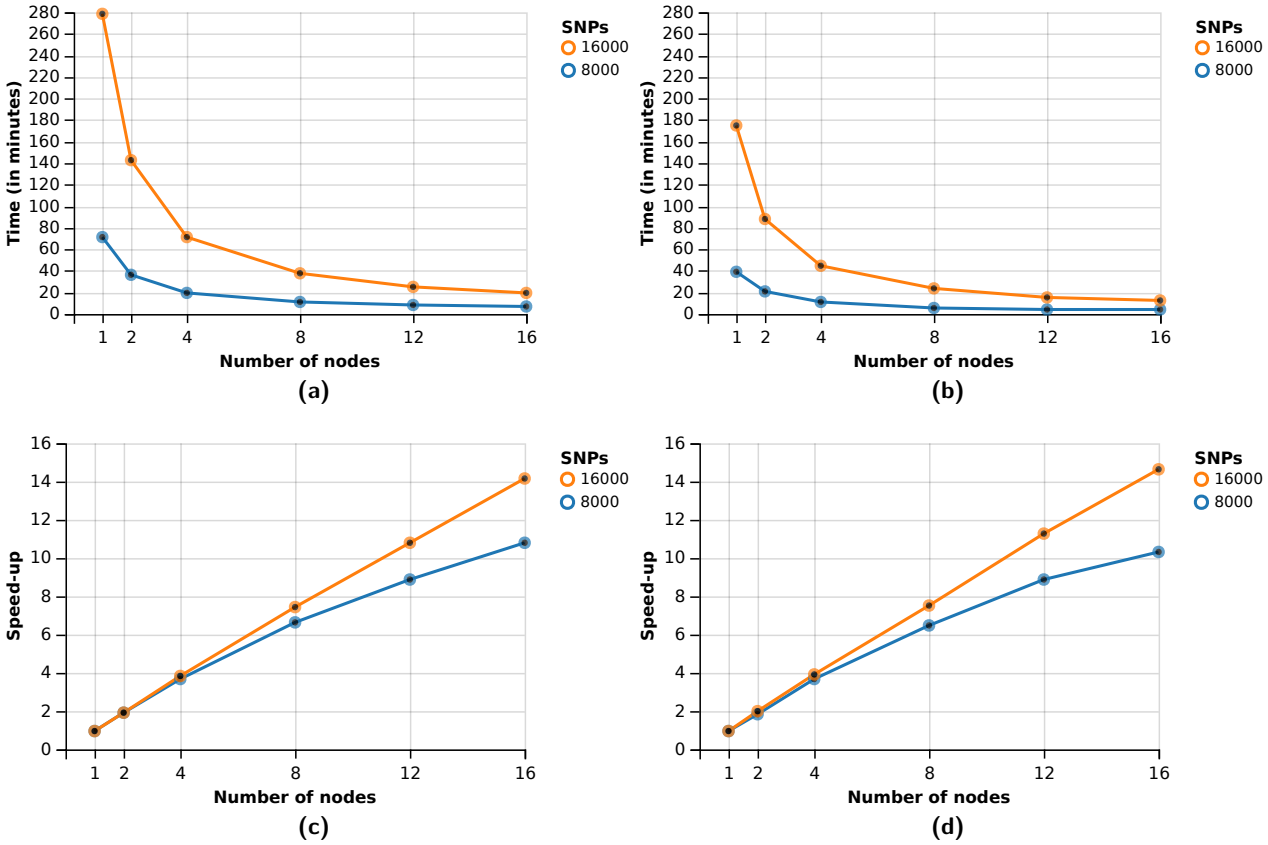


Figura 6.4: Tiempo de ejecución de la extensión para clústeres de FaST-LMM y su aceleración con diferentes números de nodos en: Piz Daint (figuras «a» y «c») y Minotauro (figuras «b» y «d»)

en ejecutarse si solo se utiliza uno de los nodos de Piz Daint. Sin embargo, y asumiendo que la eficiencia de la escalabilidad débil observada se mantiene, el problema podría ser resuelto en Piz Daint aproximadamente 19 horas usando 320 de sus nodos.

		Tiempo	
Nodos	Parejas de SNP	Piz Daint	Minotauro
1	31.996.000	4244,78	2351,77
2	63.997.641	4260,44	2402,85
4	127.992.000	4300,12	2694,75
8	255.979.251	4286,53	2684,12
12	383.991.328	4300,18	2695,05
16	511.984.000	4294,01	2715,84

Cuadro 6.3: Tiempo de ejecución (en segundos) con diferentes números de nodos fijando el tamaño de problema a 3.000 SNP por nodo en Piz Daint y en Minotauro

CAPÍTULO 7

Conclusiones generales

Este capítulo presenta las principales contribuciones y conclusiones de esta tesis doctoral. También se detallan las publicaciones, tanto en revistas como en congresos internacionales, que se han realizado durante su desarrollo.

7.1 Principales contribuciones

Las principales contribuciones de esta tesis doctoral se han centrado en dos problemas del campo de la bioinformática: el alineamiento de secuencias de ADN/ARN en un genoma de referencia y la detección de epistasis. En ambos casos, el trabajo ha consistido en el desarrollo de aplicaciones eficientes y paralelas en el marco de la computación de altas prestaciones.

La primera parte del trabajo se ha dedicado al desarrollo de un alineador de secuencias de ADN y ARN que explotan eficientemente los recursos de un sistema de computación y permite realizar el alineamiento de estas secuencias de forma rápida y precisa.

El primer alineador implementado, HPG Aligner ARN BWT, realiza el alineamiento de secuencias de ARN [36]. Utiliza la técnica de la transformada de Burrows-Wheeler para el alineamiento de las lecturas más sencillas y la de Smith-Watermam para las lecturas que no han podido ser alineadas con la primera (por tener una cantidad de diferencias con el genoma de referencia mayor que la soportada por esa técnica). Esta última técnica es mucho más sensible que la primera, pero más costosa computacionalmente.

HPG Aligner ARN BWT lleva a cabo un procesamiento por etapas donde cada etapa ejecuta un paso distinto del algoritmo. Además, estas etapas están conectadas mediante colas, que se utilizan como mecanismos de sincronización y de almacenamiento intermedio.

Para realizar este procesamiento segmentado, el primer paso del algoritmo consiste en dividir el conjunto de lecturas en paquetes de tamaño fijo. Estos paquetes atraviesan las etapas en orden hasta que la última almacena los alineamientos encontrados para cada una de las lecturas en el fichero de salida.

El paralelismo se aplica en esta implementación a dos niveles. Por un lado, el procesamiento simultáneo de las etapas del algoritmo sobre diferentes paquetes de datos de entrada y, por otro, el procesamiento simultáneo de las distintas lecturas que componen un paquete. Así pues, en el caso de que exista un número de recursos computacionales suficientemente grande (núcleos del

procesador), estos se distribuirán por grupos para que cada uno compute una etapa del algoritmo sobre distintos paquetes y, a su vez, todos los procesadores del grupo ejecutarán simultáneamente las operaciones de una etapa para las lecturas incluidas en ese paquete. Siguiendo esta aproximación, se han implementado varias versiones que se diferencian en cómo se distribuyen los núcleos del procesador entre las distintas etapas del algoritmo.

En [39] se describe una nueva implementación del alineador: HPG Aligner ARN BWT+M. Esta nueva implementación incluye las técnicas usadas hasta el momento y añade una mejora importante: la realización de varios refinamientos para poder alinear las lecturas más difíciles. Para esto, el procesamiento se ha dividido en tres flujos de trabajo consecutivos. Cada flujo mapea las lecturas que puede de forma completa y almacena las lecturas con alineamientos incompletos en ficheros temporales (que serán procesados en los flujos de trabajo siguientes). Para ayudar en el alineamiento de las lecturas se utilizan dos estructuras de datos: un metaexón y un árbol AVL. Estas estructuras almacenan la información correspondiente a las lecturas mapeadas completamente y se actualizan cada vez que se alinea completamente una lectura. La información contenida en estas estructuras en un momento dado ayudará al alineamiento de las lecturas que se procesen posteriormente.

En esta misma línea de trabajo también se ha desarrollado un nuevo método para el alineamiento de secuencias de ADN basado en la técnica de la matriz de sufijos [37], dando lugar la implementación denominada HPG Aligner ADN SA. Esta técnica también se aplicó para el procesamiento de secuencias de ARN dando lugar a HPG Aligner ARN SA+M [48]. Este último alineador se apoya en las tres características principales descritas: el procesamiento segmentado de las etapas del algoritmo, varios flujos de trabajo consecutivos y la utilización de estructuras de datos donde se almacena la información que será utilizada en los sucesivos flujos para obtener una mayor sensibilidad. Este nuevo alineador, gracias también a que trabaja sin ningún tipo de compresión del genoma de referencia, consigue una gran aceleración. Los experimentos realizados revelan la eficiencia del algoritmo sobre servidores equipados con procesadores multinúcleo, que supera a otros alineadores actuales, tanto desde el punto de vista temporal como de sensibilidad.

En [34] se presenta la versión distribuida para sistemas clúster del alineador HPG Aligner ARN BWT+M, denominado HPG Aligner ARN BWT+M+MPI. En este caso, como el procesamiento está distribuido entre los nodos del clúster, es necesario sincronizar los datos justo antes de iniciar un nuevo flujo del procesamiento. Esta nueva versión utiliza MPI para la comunicación entre los nodos del clúster y distribuye el fichero de entrada uniformemente entre estos, repartiendo la carga de trabajo de forma equilibrada en el sistema. Con esta nueva organización se consigue reducir del tiempo de ejecución al aplicar técnicas de paralelismo multinodo sin perder sensibilidad.

Finalmente, y como conclusión al trabajo realizado en este problema, en [35] se presentó un entorno genérico de trabajo, o *framework*, para el alineamiento de secuencias de ADN/ARN, que permite que cualquier alineador pueda ejecutarse en un clúster de computadores aprovechando todos sus recursos sin que sea necesario que esté programado para ello. Además, siguiendo el diseño y la operación de nuestras anteriores implementaciones, organizamos este *framework* en un flujo de trabajo de dos etapas, para que cada una de ellas pueda ejecutar un alineador distinto. Con esta configuración, por ejemplo, la primera etapa puede, idealmente, utilizar un alineador rápido, para mapear rápidamente un gran número de lecturas, mientras que la segunda puede utilizar un alineador más sensible, y probablemente más lento, para actuar como una etapa de refinamiento sobre las lecturas no mapeadas por el primer alineador. De esta forma, se obtendría un alineador con una mayor sensibilidad que la del primer alineador y más rápido que el segundo. Además, ambos alineadores se estarían ejecutando de forma paralela en un clúster (cuando no estaban necesariamente diseñados para esto).

La segunda parte de la tesis se ha enfocado en el desarrollo de un software eficiente para el estudio de la epistasis. En esta línea, el primer paso ha consistido en analizar las diferentes aplicaciones bioinformáticas disponibles actualmente que permiten realizar este análisis. De entre todas ellas, se ha seleccionado FaST-LMM, una de las aplicaciones más utilizadas para atacar este problema por su alta sensibilidad, a pesar de su alto coste computacional. La segunda parte del trabajo se ha centrado en el análisis pormenorizado del coste temporal de este software. Este estudio nos llevó a detectar sus principales cuellos de botella. En [32] se describe este análisis y las optimizaciones propuestas e implementadas para mejorar su rendimiento. Estas mejoras se han centrado en tres ejes principales: descargar sobre un procesador gráfico el cálculo matricial; aplicar una estructura de datos optimizada para almacenar las relaciones entre SNP que se obtienen durante la ejecución de la aplicación; y optimizar determinadas operaciones para reutilizar cálculos ya realizados. Con las anteriores contribuciones se ha conseguido reducir considerablemente el tiempo de procesamiento, uno de los principales problemas de FaST-LMM.

En [33] se ha desarrollado una versión de FaST-LMM que puede ejecutarse en un clúster de procesadores multinúcleo y que es capaz de repartir la carga de trabajo entre los distintos nodos del sistema. Esta versión utiliza MPI para la comunicación entre los nodos del clúster y distribuye el fichero que contiene los datos de entrada uniformemente entre todos los nodos. Esta implementación incluye una mejora respecto a la parte que descarga a la GPU el cálculo matricial incluido en el algoritmo. Esta nueva mejora consiste en una versión multi-GPU que permite utilizar las diferentes GPU que tengan conectadas los nodo del clúster.

7.2 Publicaciones

Las contribuciones científicas de esta tesis han sido validadas en diferentes publicaciones internacionales. A continuación se presentan las publicadas en revistas internacionales y, seguidamente, las publicadas en congresos internacionales.

7.2.1 Revistas internacionales

Esta tesis ha dado lugar a las siguientes publicaciones en artículos internacionales:

1. **Acceleration of short and long DNA read mapping without loss of accuracy using suffix array** [48]
   Bioinformatics (2014), pp. 3396–3398
   Joaquín Tárraga, Vicente Arnau, **Héctor Martínez**, Raúl Moreno, Diego Cazorla, José Salavert-Torres, Ignacio Blanquer-Espert, Joaquín Dopazo e Ignacio Medina
   Esta revista está indexada por el Journal Citation Reports (JCR) de 2014 en la posición 19 de 163 (Q1) de la categoría Biotechnology & Applied Microbiology

2. **Concurrent and Accurate Short Read Mapping on Multicore Processors** [37]
   IEEE/ACM Transactions on Computational Biology and Bioinformatics (2015), pp. 995–1007
   Héctor Martínez, Joaquín Tárraga, Ignacio Medina, Sergio Barrachina, Maribel Castillo, Joaquín Dopazo y Enrique S. Quintana-Ortí
   Esta revista está indexada por el Journal Citation Reports (JCR) de 2015 en la posición 26 de 123 (Q1) de la categoría Statistics & Probability.

3. **Highly sensitive and ultrafast read mapping for RNA-seq analysis** [39]
   DNA Research (2016), pp. 93–100

Ignacio Medina, Joaquín Tárraga, **Héctor Martínez**, Sergio Barrachina, Maribel Castillo, J. Paschall, José Salavert-Torres, Ignacio Blanquer-Espert, V. Hernández-García, Enrique S. Quintana-Ortí y Joaquín Dopazo
Esta revista está indexada por el Journal Citation Reports (JCR) de 2016 en la posición 22 de 167 (Q1) de la categoría Genetics & Heredity.

4. **A framework for genomic sequencing on clusters of multicore and manycore processors** [35]
The International Journal of High Performance Computing Applications (2018), pp. 393–406
Héctor Martínez, Sergio Barrachina, Maribel Castillo, Joaquín Tárraga, Ignacio Medina, Joaquín Dopazo y Enrique S. Quintana-Ortí
Esta revista está indexada por el Journal Citation Reports (JCR) de 2018 en la posición 39 de 104 (Q2) de la categoría Computer science, theory & methods.

5. **FaST-LMM for Two-Way Epistasis Tests on High-Performance Clusters** [33]
Journal of Computational Biology (2018), pp. 862–870
Héctor Martínez, Sergio Barrachina, Maribel Castillo, Enrique S. Quintana-Ortí, Jordi Rambla de Argila, Xavier Farré y Arcadi Navarro
Esta revista está indexada por el Journal Citation Reports (JCR) de 2018 en la posición 76 de 123 (Q3) de la categoría Statistics & Probability.

6. **Dynamic reconfiguration of noniterative scientific applications: A case study with HPG aligner** [20]
The International Journal of High Performance Computing Applications (2018)
Sergio Iserte, **Héctor Martínez**, Sergio Barrachina, Maribel Castillo, Rafael Mayo y Antonio J. Peña
Esta revista está indexada por el Journal Citation Reports (JCR) de 2018 en la posición 39 de 104 (Q2) de la categoría Computer science, theory & methods.

7.2.2 Congresos internacionales

Esta tesis ha dado lugar a las siguientes publicaciones en congresos internacionales:

1. **A Dynamic Pipeline for RNA Sequencing on Multicore Processors** [36]
Proceedings of the 20th European MPI Users' Group Meeting (EuroMPI '13) (2013), pp. 235–240
Héctor Martínez, Joaquín Tárraga, Ignacio Medina, Sergio Barrachina, Maribel Castillo, Joaquín Dopazo y Enrique S. Quintana-Ortí
Este congreso está clasificado como CORE:C y SHINE:B en el GII-GRIN Conference Rating 2013.

2. **Scalable RNA Sequencing on Clusters of Multicore Processors** [34]
14th IEEE International Conference on Trust, Security and Privacy in Computing and Communications, TrustCom 2015 (2015), pp. 190–195
Héctor Martínez, Sergio Barrachina, Maribel Castillo, Joaquín Tarraga, Ignacio Medina, Joaquín Dopazo y Enrique S. Quintana-Ortí
Este congreso está clasificado como CORE:B, MAS:C, SHINE:C según el GII-GRIN Conference Rating 2015.

3. **Accelerating FaST-LMM for Epistasis Tests** [32]
Actas de International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP)

(2017), pp. 548–557
Héctor Martínez, Sergio Barrachina, Maribel Castillo, Enrique S. Quintana-Ortí, Jordi Rambla de Argila, Xavier Farré y Arcadi Navarro
Este congreso está clasificado según el GII-GRIN-SCIE (GGS) Conference Rating 2017 como CORE:B, MAS:C, SHINE:C.

7.3 Conclusiones

Respecto a la primera parte del trabajo se puede concluir que el alineamiento de secuencias de ADN/ARN en un genoma, debido a su importancia dentro del campo de la bioinformática, ha sufrido una gran evolución en estos últimos años. Por esta razón, se están desarrollando nuevas aplicaciones que usan técnicas de mapeo más eficientes y que son, por tanto, capaces de alinear un mayor número de lecturas en menos tiempo y con mayor sensibilidad. Por otro lado, debido a las nuevas técnicas de secuenciación así como a las técnicas de mapeo utilizadas, los algoritmos de secuenciación, cada vez más, necesitan también sistemas con una gran capacidad de almacenamiento. Por ello, las aplicaciones propuestas en esta tesis se han desarrollado con el objetivo de conseguir la mayor sensibilidad posible y, a la vez, para que puedan aprovechar todos los recursos de almacenamiento y procesamiento a su disposición, ya sea en grandes sistemas multiprocesador o en clústeres de computadores.

En el caso del alineamiento de secuencias de ADN, el mayor problema vienen dado por la gran cantidad de errores que puede contener una lectura debido a múltiples factores. Por esta razón, el desarrollo de un alineador para secuencias de ADN presenta una gran dificultad. Sin embargo, gracias a las diferentes técnicas algorítmicas con una gran sensibilidad aplicadas, tales como la matriz de sufijos junto con la tabla LCP y la técnica de compresión de matrices CRS, se ha conseguido obtener mejores resultados que los de los alineadores actuales.

En el caso del alineamiento de secuencias de ARN, el mayor problema consiste en la detección de los puntos de unión entre exones, además de en los errores que se pueden concentrar en una lectura. Como se puede observar en los resultados obtenidos, el alineador desarrollado es capaz de superar la sensibilidad de los actuales incluso incrementado su velocidad de procesamiento. Esto ha sido posible gracias a las diferentes técnicas algorítmicas aplicadas para realizar los mapeos y, también, a la utilización de distintas estructuras de datos, tales como el metaexón o el árbol AVL, que permiten refinar los alineamientos a partir de la información obtenida de los alineamientos realizados previamente.

En relación a la segunda parte del trabajo, debido a la gran cantidad de aplicaciones existentes para el estudio de la epistasis, se decidió, en lugar de desarrollar una nueva aplicación, evaluar las ya existentes con el fin de seleccionar una de ellas y ver si era posible optimizarla y, en el caso de que no pudiera ejecutarse sobre clústeres, dotarle de esta funcionalidad. La aplicación que finalmente se eligió, gracias a la ayuda de expertos en el campo de la genómica, y debido a su gran sensibilidad, fue FaST-LMM. Sin embargo, su gran sensibilidad lleva aparejada un mayor coste computacional que el de otras aplicaciones de epistasis. Así, su uso se hace prácticamente inviable para el procesamiento de ficheros de entrada de tamaño moderado (500.000 SNP). Como parte de esta tesis, se ha realizado un estudio detallado de su rendimiento y de los cuellos de botella que presentaba. A raíz de este estudio, se ha desarrollado una versión que reduce notablemente su coste temporal. También se ha implementado una aplicación paralela para clústeres a partir de la versión optimizada para un único nodo. Esta versión permite distribuir el fichero de entrada entre los distintos nodos del sistema y, de esta forma, trabajar con ficheros mucho mayores reduciendo el tiempo de cómputo y manteniendo la precisión.

En general, se puede concluir que, en las distintas implementaciones realizadas a lo largo de este trabajo, se han aplicado diferentes técnicas HPC con dos objetivos fundamentales: incrementar la velocidad de procesamiento y mantener o incrementar la sensibilidad de los resultados. Se han desarrollado aplicaciones paralelas que han permitido explotar el paralelismo a nivel de hilos de ejecución, en sistemas multinúcleo, y a nivel de procesos, en clústeres. En este último caso también se ha conseguido aumentar la capacidad de almacenamiento del sistema al disponer de varios nodos. Por otro lado, también se ha explotado el uso de aceleradores, como la GPU, que permiten explotar el paralelismo a gran escala en operaciones que requieren de gran capacidad de cálculo, descargando operaciones matriciales con elevado coste computacional a este tipo de aceleradores.

Desde un punto de vista más personal, este trabajo ha supuesto un primer paso de estudio para formarme en los dos temas que han sido el objetivo de este trabajo. Ambos relacionados con dos problemas muy actuales de la bioinformática y que desconocía completamente. El trabajo realizado siempre se ha hecho en colaboración con grupos de investigación expertos en estos temas, lo que nos ha ayudado a centrar las ideas para poder realizar los desarrollos posteriores.

Como conclusión final, pensamos que el trabajo desempeñado ha contribuido de forma positiva a la comunidad científica de la rama de la biología, puesto que se han desarrollado aplicaciones que les ayudará en su tarea notablemente, tanto desde el punto de vista del coste temporal como de los resultados obtenidos en los análisis.

7.4 Líneas abiertas de investigación

Como se ha comentado, la continua evolución de las tecnologías de secuenciación obliga a que el software para el estudio de las secuencias de ADN/ARN no cese de mejorar. Esto significa que, aunque se hayan conseguido grandes mejoras con el software desarrollado, hay que continuar mejorándolo e incorporar nuevas técnicas de programación para acelerar su procesamiento.

En concreto, los alineadores desarrollados no usan ningún tipo de dispositivo GPU, y por lo tanto, hacerlo supondría una posible mejora futura. Esto permitiría descargar el trabajo de la CPU a la GPU y acelerar el procesamiento. Por otro lado, otra posible mejora sería incorporar la siguiente parte del estudio (que consiste en el estudio de las variantes encontradas) a la secuenciación. Esto permitiría, por una parte, devolver las variantes encontradas a la vez que los alineamientos, lo que ahorraría a los biólogos tener que realizar este paso. Por otro parte, sería posible solapar ambos procesamientos, lo que podría reducir el tiempo necesario para encontrar las variantes.

Respecto al estudio de la epistasis, existen muchas líneas de investigación abiertas. Esto es debido a que la mayor parte de los algoritmos actuales únicamente realizan estudios de 2 y 3vías. Desarrollar un nuevo algoritmo con un número de vías mayor, implicaría un gran avance en este tipo de estudio genómico, ya que permitiría encontrar relaciones entre un mayor número de variantes genómicas.

7.5 Trabajo futuro

Tras la realización de este trabajo se han abierto varias líneas de investigación. Por un lado, en el ámbito de la computación paralela, un aspecto que está alcanzando importancia es el de la maleabilidad de las aplicaciones de forma que, en determinados momentos de la ejecución, sea posible modificar la asignación de recursos. No se conocen trabajos previos en el ámbito de la bioinformática que impliquen la utilización de técnicas de computación paralela con capacidad de maleabilidad. En esta dirección se ha realizado una primera aproximación en [20]. En este trabajo se estudió la posibilidad de utilizar DMR API para generar una versión maleable de HPG Aligner ARN

BWT+M MPI, el alineador descrito anteriormente para clústeres de computadores con memoria distribuida. Esta aplicación es no iterativa y presenta un patrón de comunicación irregular entre los distintos procesos. Esta primera aproximación nos ha permitido probar los beneficios de este tipo de aplicaciones en clústeres de producción. Los resultados experimentales muestran un reducción importante del tiempo de finalización de trabajos de HPG Aligner ARN BWT+M MPI maleable en comparación con el requerido por HPG Aligner ARN BWT+M MPI, cuando se considera una carga de trabajo determinada, además de conseguir un rendimiento mayor del sistema.

Por otro lado, también hemos comprobado que la utilización de la computación de altas prestaciones en bioinformática es fundamental desde la aparición de los nuevos secuenciadores. La enorme cantidad de datos generados no se puede tratar en un periodo de tiempo asumible si no se emplean herramientas de cómputo paralelo. Un estudio genómico desde el ámbito de la bioinformática está compuesto por dos fases principales. La primera obtiene las secuencias de bases de nucleótidos de la muestra a analizar. La segunda realiza el análisis y la extracción de información relevante a partir de estas secuencias mediante herramientas informáticas específicas. En esta parte es donde puede situarse el trabajo futuro propuesto a partir de la realización de esta tesis. Todo este análisis se pueden incluir en un entorno de trabajo donde el usuario final únicamente tenga que seleccionar los estudios a realizar e indicar los ficheros que contienen los datos de entrada y donde se deberán almacenar los resultados. Actualmente se está en contacto con un grupo de investigación del Hospital General de Castellón para iniciar una colaboración y solicitar un proyecto conjunto en esta línea. El objetivo del proyecto es el desarrollo de una herramienta informática que, tras la cirugía de citorreducción primaria de un cáncer de ovario, permita precisar la resección óptima basándose en la presencia de ADNct en plasma. En este proyecto se utilizarían algunas de las aplicaciones desarrolladas en esta tesis, se desarrollarían otras nuevas y se complementaría con otras aplicaciones ya existentes, de tal forma que fuera posible realizar el análisis requerido.

Bibliografía

[1] Abraham, G., Tye-Din, J. A., Bhalala, O. G., Kowalczyk, A., Zobel, J., and Inouye, M. Accurate and Robust Genomic Prediction of Celiac Disease Using Statistical Learning. *PLOS Genetics 10*, 2 (feb 2014), e1004137.

[2] Adzhubei, I., Jordan, D. M., and Sunyaev, S. R. Predicting functional effect of human missense mutations using PolyPhen-2. *Current protocols in human genetics Chapter 7* (jan 2013), Unit7.20–Unit7.20.

[3] Biesecker, L. G. Exome sequencing makes medical genomics a reality. *Nature Genetics 42* (jan 2010), 13.

[4] Biesecker, L. G. Opportunities and challenges for the integration of massively parallel genomic sequencing into clinical practice: lessons from the ClinSeq project. *Genetics In Medicine 14* (feb 2012), 393.

[5] Burrows, M., and J. Wheeler, D. *A Block-Sorting Lossless Data Compression Algorithm*, vol. 1. jul 1995.

[6] Chaisson, M. J., and Tesler, G. Mapping single molecule sequencing reads using basic local alignment with successive refinement (BLASR): application and theory. *BMC Bioinformatics 13*, 1 (2012), 238.

[7] Chhangawala, S., Rudy, G., Mason, C. E., and Rosenfeld, J. A. The impact of read length on quantification of differentially expressed genes and splice junction detection. *Genome Biology 16*, 1 (2015), 131.

[8] Cingolani, P., Platts, A., Wang, L. L., Coon, M., Nguyen, T., Wang, L., Land, S. J., Lu, X., and Ruden, D. M. A program for annotating and predicting the effects of single nucleotide polymorphisms, SnpEff: SNPs in the genome of Drosophila melanogaster strain w1118; iso-2; iso-3. *Fly 6*, 2 (apr 2012), 80–92.

[9] Cock, P. J. A., Fields, C. J., Goto, N., Heuer, M. L., and Rice, P. M. The Sanger FASTQ file format for sequences with quality scores, and the Solexa/Illumina FASTQ variants. *Nucleic acids research 38*, 6 (apr 2010), 1767–1771.

[10] Coleman, C., Quinn, E. M., Ryan, A. W., Conroy, J., Trimble, V., Mahmud, N., Kennedy, N., Corvin, A. P., Morris, D. W., Donohoe, G., O'Morain, C., MacMathuna, P., Byrnes, V., Kiat, C., Trynka, G., Wijmenga, C., Kelleher, D., Ennis, S., Anney, R. J. L., and McManus, R. Common polygenic variation in coeliac disease and confirmation of ZNF335 and NIFA as disease susceptibility loci. *European journal of human genetics : EJHG 24*, 2 (feb 2016), 291–297.

[11] DeLuca, D. S., Levin, J. Z., Sivachenko, A., Fennell, T., Nazaire, M.-D., Williams, C., Reich, M., Winckler, W., and Getz, G. RNA-SeQC: RNA-seq metrics for quality control and process optimization. *Bioinformatics (Oxford, England) 28*, 11 (jun 2012), 1530–1532.

[12] Dobin, A., Davis, C. A., Schlesinger, F., Drenkow, J., Zaleski, C., Jha, S., Batut, P., Chaisson, M., and Gingeras, T. R. STAR: ultrafast universal RNA-seq aligner. *Bioinformatics 29*, 1 (jan 2013), 15–21.

[13] Ferragina, P., and Manzini, G. Opportunistic data structures with applications. In *Proceedings 41st Annual Symposium on Foundations of Computer Science* (2000), pp. 390–398.

[14] Fullwood, M. J., Wei, C.-L., Liu, E. T., and Ruan, Y. Next-generation DNA sequencing of paired-end tags (PET) for transcriptome and genome analyses. *Genome research 19*, 4 (apr 2009), 521–532.

[15] González-Domínguez, J., Schmidt, B., Kässens, J. C., and Wienbrandt, L. Hybrid CPU/GPU Acceleration of Detection of 2-SNP Epistatic Interactions in GWAS BT - Euro-Par 2014 Parallel Processing. F. Silva, I. Dutra, and V. Santos Costa, Eds., Springer International Publishing, pp. 680–691.

[16] Grant, G. R., Farkas, M. H., Pizarro, A. D., Lahens, N. F., Schug, J., Brunk, B. P., Stoeckert, C. J., Hogenesch, J. B., and Pierce, E. A. Comparative analysis of RNA-Seq alignment algorithms and the RNA-Seq unified mapper (RUM). *Bioinformatics (Oxford, England) 27*, 18 (sep 2011), 2518–2528.

[17] Hemani, G., Shakhbazov, K., Westra, H.-J., Esko, T., Henders, A. K., McRae, A. F., Yang, J., Gibson, G., Martin, N. G., Metspalu, A., Franke, L., Montgomery, G. W., Visscher, P. M., and Powell, J. E. Detection and replication of epistasis influencing transcription in humans. *Nature 508*, 7495 (apr 2014), 249–253.

[18] Hemani, G., Theocharidis, A., Wei, W., and Haley, C. EpiGPU: exhaustive pairwise epistasis scans parallelized on consumer level graphics cards. *Bioinformatics 27*, 11 (jun 2011), 1462–1465.

[19] Hood, L., and Rowen, L. The Human Genome Project: big science transforms biology and medicine. *Genome medicine 5*, 9 (sep 2013), 79.

[20] Iserte, S., Martínez, H., Barrachina, S., Castillo, M., Mayo, R., and Peña, A. J. Dynamic reconfiguration of noniterative scientific applications: A case study with HPG aligner. *The International Journal of High Performance Computing Applications* (sep 2018), 1094342018802347.

[21] Jiang, G., Ma, Y., An, T., Pan, Y., Mo, F., Zhao, D., Liu, Y., Miao, J.-N., Gu, Y.-J., Wang, Y., and Gao, S.-H. Relationships of circular RNA with diabetes and depression. *Scientific reports 7*, 1 (aug 2017), 7285.

[22] Kim, D., Langmead, B., and Salzberg, S. L. HISAT: a fast spliced aligner with low memory requirements. *Nature methods 12*, 4 (apr 2015), 357–360.

[23] Langmead, B., Trapnell, C., Pop, M., and Salzberg, S. L. Ultrafast and memory-efficient alignment of short DNA sequences to the human genome. *Genome Biology 10*, 3 (2009), R25.

[24] Lappalainen, I., Almeida-King, J., Kumanduri, V., Senf, A., Spalding, J. D., Ur-Rehman, S., Saunders, G., Kandasamy, J., Caccamo, M., Leinonen, R., Vaughan, B., Laurent, T., Rowland, F., Marin-Garcia, P., Barker, J., Jokinen, P., Torres, A. C., de Argila, J. R., Llobet, O. M., Medina, I., Puy, M. S., Alberich, M., de la Torre, S., Navarro, A., Paschall, J., and Flicek, P. The European Genome-phenome Archive of human data consented for biomedical research. *Nature genetics 47*, 7 (jul 2015), 692–695.

[25] Li, H. *Aligning sequence reads, clone sequences and assembly contigs with BWA-MEM*, vol. 1303. mar 2013.

[26] Li, H., Handsaker, B., Wysoker, A., Fennell, T., Ruan, J., Homer, N., Marth, G., Abecasis, G., Durbin, R., and Subgroup, . G. P. D. P. The Sequence Alignment/Map format and SAMtools. *Bioinformatics (Oxford, England) 25*, 16 (aug 2009), 2078–2079.

[27] Lippert, C., Listgarten, J., Davidson, R. I., Baxter, J., Poon, H., Kadie, C. M., and Heckerman, D. An Exhaustive Epistatic SNP Association Analysis on Expanded Wellcome Trust Data. *Scientific Reports 3* (jan 2013), 1099.

[28] Liu, C.-M., Wong, T., Wu, E., Luo, R., Yiu, S.-M., Li, Y., Wang, B., Yu, C., Chu, X., Zhao, K., Li, R., and Lam, T.-W. SOAP3: ultra-fast GPU-based parallel alignment tool for short reads. *Bioinformatics 28*, 6 (mar 2012), 878–879.

[29] Love, M. I., Anders, S., Kim, V., and Huber, W. RNA-Seq workflow: gene-level exploratory analysis and differential expression. *F1000Research 4* (oct 2015), 1070.

[30] Manber, U., and Myers, G. Suffix arrays: A new method for on-line string searches. In *Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms* (Philadelphia, PA, USA, 1990), SODA '90, Society for Industrial and Applied Mathematics, pp. 319–327.

[31] Manber, U., and Myers, G. Suffix Arrays: A New Method for On-Line String Searches. *SIAM Journal on Computing 22*, 5 (oct 1993), 935–948.

[32] Martínez, H., Barrachina, S., Castillo, M., Quintana-Ortí, E. S., De Argila, J. R., Farré, X., and Navarro, A. Accelerating FaST-LMM for Epistasis Tests BT - Algorithms and Architectures for Parallel Processing. S. Ibrahim, K.-K. R. Choo, Z. Yan, and W. Pedrycz, Eds., Springer International Publishing, pp. 548–557.

[33] Martínez, H., Barrachina, S., Castillo, M., Quintana-Ortí, E. S., Rambla de Argila, J., Farré, X., and Navarro, A. FaST-LMM for Two-Way Epistasis Tests on High-Performance Clusters. *Journal of Computational Biology 25*, 8 (jul 2018), 862–870.

[34] Martinez, H., Barrachina, S., Castillo, M., Tarraga, J., Medina, I., Dopazo, J., and Quintana-Orti, E. S. Scalable RNA Sequencing on Clusters of Multicore Processors. In *Proceedings - 14th IEEE International Conference on Trust, Security and Privacy in Computing and Communications, TrustCom 2015* (2015), vol. 3, pp. 190–195.

[35] Martínez, H., Barrachina, S., Castillo, M., Tárraga, J., Medina, I., Dopazo, J., and Quintana-Ortí, E. S. A framework for genomic sequencing on clusters of multicore and manycore processors. *The International Journal of High Performance Computing Applications 32*, 3 (jun 2018), 393–406.

[36] Martínez, H., Tárraga, J., Medina, I., Barrachina, S., Castillo, M., Dopazo, J., and Quintana-Ortí, E. S. A Dynamic Pipeline for RNA Sequencing on Multicore Processors. In *Proceedings of the 20th European MPI Users' Group Meeting* (New York, NY, USA, 2013), EuroMPI '13, ACM, pp. 235–240.

[37] Martínez, H., Tárraga, J., Medina, I., Barrachina, S., Castillo, M., Dopazo, J., and Quintana-Ortí, E. S. Concurrent and Accurate Short Read Mapping on Multicore Processors. *IEEE/ACM Transactions on Computational Biology and Bioinformatics 12*, 5 (2015), 995–1007.

[38] Martínez, H., Tárraga, J., Medina, I., Barrachina, S., Castillo, M., Dopazo, J., and Quintana-Ortí, E. S. Concurrent and accurate RNA sequencing on multicore platforms. Tech. Rep. ICC 2013-03-01, Departamento de Ingeniería y Ciencia de los Computadores, Universidad Jaume I, Spain, 2013.

[39] Medina, I., Tárraga, J., Martínez, H., Barrachina, S., Castillo, M. I., Paschall, J., Salavert-Torres, J., Blanquer-Espert, I., Hernández-García, V., Quintana-Ortí, E. S., and Dopazo, J. Highly sensitive and ultrafast read mapping for RNA-seq analysis. *DNA Research 23*, 2 (apr 2016), 93–100.

[40] Mishima, H., Sasaki, K., Tanaka, M., Tatebe, O., and Yoshiura, K.-I. Agile parallel bioinformatics workflow management using Pwrake. *BMC research notes 4* (sep 2011), 331.

[41] Mitchell, L. The cambridge dictionary of statistics (third edition) Everitt BS (ed.) (2006) ISBN: 0521690277; 432 pages; £17.99, $40.50 Cambridge University Press; http://www.cambridge.org/. *Pharmaceutical Statistics 6*, 1 (feb 2007), 71–72.

[42] Nair, P. S., and Vihinen, M. VariBench: A Benchmark Database for Variations. *Human Mutation 34*, 1 (jan 2013), 42–49.

[43] Ng, P. C., and Henikoff, S. SIFT: Predicting amino acid changes that affect protein function. *Nucleic acids research 31*, 13 (jul 2003), 3812–3814.

[44] Raczy, C., Petrovski, R., Saunders, C. T., Chorny, I., Kruglyak, S., Margulies, E. H., Chuang, H. Y., Källberg, M., Kumar, S. A., Liao, A., Little, K. M., Strömberg, M. P., and Tanner, S. W. Isaac: Ultra-fast whole-genome secondary analysis on Illumina sequencing platforms. *Bioinformatics 29*, 16 (aug 2013), 2041–2043.

[45] Schwanhäusser, B., Busse, D., Li, N., Dittmar, G., Schuchhardt, J., Wolf, J., Chen, W., and Selbach, M. Global quantification of mammalian gene expression control. *Nature 473* (may 2011), 337.

[46] Smith, T., and Waterman, M. Identification of common molecular subsequences. *Journal of Molecular Biology 147*, 1 (mar 1981), 195–197.

[47] Stephens, Z. D., Lee, S. Y., Faghri, F., Campbell, R. H., Zhai, C., Efron, M. J., Iyer, R., Schatz, M. C., Sinha, S., and Robinson, G. E. Big Data: Astronomical or Genomical? *PLOS Biology 13*, 7 (2015), 1–11.

[48] Tarraga, J., Arnau, V., Martinez, H., Moreno, R., Cazorla, D., Salavert-Torres, J., Blanquer-Espert, I., Dopazo, J., and Medina, I. Acceleration of short and long DNA read mapping without loss of accuracy using suffix array. *Bioinformatics 30*, 23 (dec 2014), 3396–3398.

[49] Torres, J. S., Espert, I. B., Dominguez, A. T., Hernendez, V., Medina, I., Terraga, J., and Dopazo, J. Using GPUs for the Exact Alignment of Short-Read Genetic Sequences by Means of the Burrows-Wheeler Transform. *IEEE/ACM Transactions on Computational Biology and Bioinformatics 9*, 4 (2012), 1245–1256.

[50] Trapnell, C., Pachter, L., and Salzberg, S. L. TopHat: discovering splice junctions with RNA-Seq. *Bioinformatics (Oxford, England) 25*, 9 (may 2009), 1105–1111.

[51] UCSC Genome Bioinformatics. BED format. `http://genome.ucsc.edu/FAQ/FAQformat.html#format1`.

[52] Wan, X., Yang, C., Yang, Q., Xue, H., Fan, X., Tang, N. L. S., and Yu, W. BOOST: A fast approach to detecting gene-gene interactions in genome-wide case-control studies. *American journal of human genetics 87*, 3 (sep 2010), 325–340.

[53] Wan, X., Yang, C., Yang, Q., Xue, H., Tang, N. L. S., and Yu, W. Predictive rule inference for epistatic interaction detection in genome-wide association studies. *Bioinformatics 26*, 1 (oct 2009), 30–37.

[54] Wang, K., Singh, D., Zeng, Z., Coleman, S. J., Huang, Y., Savich, G. L., He, X., Mieczkowski, P., Grimm, S. A., Perou, C. M., MacLeod, J. N., Chiang, D. Y., Prins, J. F., and Liu, J. MapSplice: accurate mapping of RNA-seq reads for splice junction discovery. *Nucleic acids research 38*, 18 (oct 2010), e178–e178.

[55] WATSON, J. D., and CRICK, F. H. C. Molecular Structure of Nucleic Acids: A Structure for Deoxyribose Nucleic Acid. *Nature 171*, 4356 (1953), 737–738.

[56] Ye, K., Schulz, M. H., Long, Q., Apweiler, R., and Ning, Z. Pindel: a pattern growth approach to detect break points of large deletions and medium sized insertions from paired-end short reads. *Bioinformatics (Oxford, England) 25*, 21 (nov 2009), 2865–2871.

[57] Yung, L. S., Yang, C., Wan, X., and Yu, W. GBOOST: a GPU-based tool for detecting gene-gene interactions in genome-wide case control studies. *Bioinformatics (Oxford, England) 27*, 9 (may 2011), 1309–1310.

[58] Zhang, X., Huang, S., Zou, F., and Wang, W. TEAM: efficient two-locus epistasis tests in human genome-wide association study. *Bioinformatics (Oxford, England) 26*, 12 (jun 2010), i217–i227.

[59] Zuk, O., Hechter, E., Sunyaev, S. R., and Lander, E. S. The mystery of missing heritability: Genetic interactions create phantom heritability. *Proceedings of the National Academy of Sciences of the United States of America 109*, 4 (jan 2012), 1193–1198.